Tugas Besar 1 IF2211 Strategi Algoritma

Pemanfaatan Algoritma *Greedy* dalam Aplikasi Permainan "*Worms*"

Oleh:

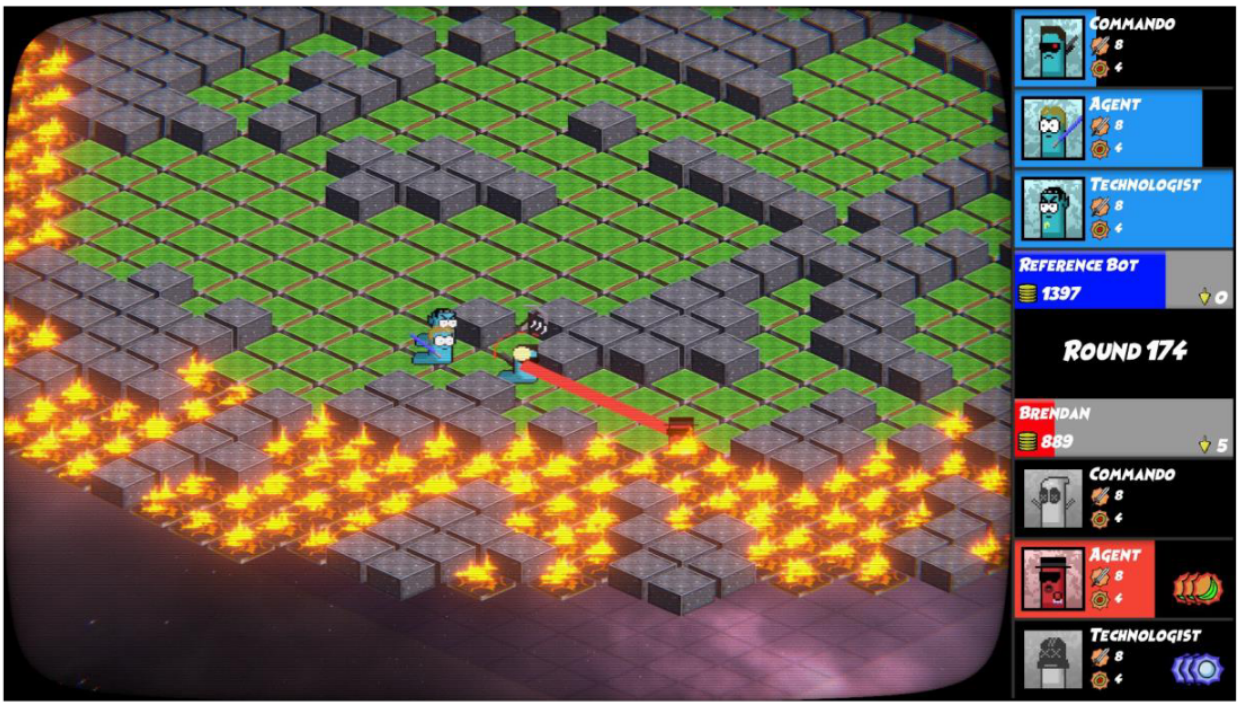
Raffi Zulvian Muzhaffar 13519003
Muhammad Fahkry Malta 13519032
Prana Gusriana 13519195

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

Daftar Isi

BAB 1
Deskripsi

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. Contoh tampilan permainan *Worms*

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.

BAB 2
Landasan Teori

2. 1 Algoritma *Greedy*

Algoritma *Greedy* merupakan metode atau algoritma yang populer dan sederhana untuk menyelesaikan persoalan optimasi. Persoalan optimasi (*optimization problems*) merupakan persoalan pencarian solusi yang optimal dan hanya ada dua macam persoalan, yaitu maksimasi (*maximization*) dan minimasi (*minimization*). Kata "*greedy*" berarti rakus atau tamak sesuai dengan prinsip dari algoritma ini yaitu "*take what you can get now!*" yang berarti ambil apapun yang bisa kamu ambil sekarang sehingga algoritma *greedy* ini membentuk solusi langkah per langkah (*step by step*). Pada setiap langkah penyelesaian masalah, terdapat banyak pilihan yang perlu dievaluasi. Oleh sebab itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan karena tidak bisa mundur kembali ke langkah sebelumnya. Jadi pada setiap langkah, kita memilih optimum lokal (*local optimum*) dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (*global optimum*). Namun, optimum global belum tentu merupakan solusi yang terbaik dan bisa jadi merupakan solusi sub-optimum atau *pseudo-optimum*.

Ada elemen-elemen yang harus diperhatikan dalam penggunaan algoritma Greedy untuk menyelesaikan masalah. Elemen tersebut antara lain adalah himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Himpunan kandidat berisi kandidat yang akan dipilih pada setiap langkah. Himpunan solusi berisi kandidat yang sudah dipilih. Fungsi solusi merupakan fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberi solusi. Fungsi seleksi merupakan fungsi untuk memilih kandidat berdasarkan strategi greedy tertentu. Fungsi kelayakan merupakan fungsi untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak). Fungsi objektif merupakan fungsi untuk memaksimumkan atau meminimumkan.

Secara umum, algoritma Greedy melibatkan pencarian sebuah himpunan bagian (himpunan solusi) dari himpunan kandidat yang dalam hal ini himpunan solusi tersebut harus memenuhi beberapa kriteria yang ditentukan oleh fungsi seleksi, fungsi kelayakan, dan fungsi solusi. Sehingga himpunan solusi menyatakan suatu solusi dan himpunan solusi dioptimasi oleh fungsi objektif.

2. 2 Pemanfaatan Game Engine

Petunjuk dalam pemanfaatan game engine adalah sebagai berikut.

1. Untuk menambahkan pemain dapat dilakukan dengan mengedit player-a dan player-b dengan bot yang anda inginkan pada file “game-runner-config.json”, Anda juga dapat mengubah file “bot.json” untuk mengatur informasi terkait bot anda.
2. Untuk menambahkan strategi *greedy* pada *game engine* maka Anda dapat memodifikasi bot yang disediakan di starter bot.
3. Untuk menjalankan permainan, Anda dapat membuka file “run.bat” (Untuk Windows/Mac dapat buka dengan double-click, Untuk Linux dapat menjalankan command “make run”).

BAB 3
Pemanfaatan Algoritma Greedy

3.1 Proses Mapping Persoalan Worms

Inti dari permainan Worms ini adalah setiap rondenya kita harus memilih cacing tim mana untuk melakukan *command* apa sesuai dengan pertimbangan tertentu sehingga pilihan tersebut diharapkan akan membuat kita memenangkan permainan. Berdasarkan hal tersebut dapat dikatakan bahwa permainan Worms ini merupakan persoalan optimasi dalam hal memilih *command* yang tepat sehingga strategi *greedy* dapat diimplementasikan pada permainan ini. Dalam menyelesaikan permasalahan dengan strategi *greedy* terdapat beberapa hal yang harus diperhatikan, yaitu himpunan kandidat, himpunan solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif.

Himpunan kandidat berisi kandidat yang akan dipilih pada setiap langkahnya. Pada permainan Worms ini pada setiap rondenya akan dipilih cacing mana untuk melakukan suatu *command* dengan pertimbangan tertentu sehingga himpunan kandidatnya adalah seluruh cacing yang ada pada permainan serta *command-command* yang tersedia pada permainan ini, yaitu *move*, *dig*, *shoot*, *do nothing*, *banana bomb*, *snowball*, dan *select*. Himpunan solusi berisi kandidat yang sudah dipilih sehingga himpunan solusi pada permainan ini adalah salah satu *command* yang akan dijalankan oleh salah satu cacing tim dan telah diseleksi dengan fungsi seleksi dan fungsi kelayakan. Fungsi seleksi merupakan fungsi untuk memilih kandidat berdasarkan strategi greedy tertentu dan pada pengimplementasian permainan Worms ini kami membuat fungsi seleksi dengan strategi menargetkan musuh dengan jarak terdekat atau menargetkan musuh dengan *health power* paling rendah dan strategi penggunaan senjata dengan urutan prioritas dari damage yang paling tinggi (*banana bomb, snowball, basic weapon*). Fungsi kelayakan merupakan fungsi untuk memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi atau tidak. Fungsi kelayakan yang kami buat adalah untuk memeriksa apakah *command* yang akan dipilih sudah memenuhi syarat seperti penggunaan senjata *banana bomb* dan *snowball* yang tidak melebihi jumlah yang ditentukan, penggunaan *select command* yang tidak melebihi jumlah yang ditentukan, *type* dari *cell* untuk menentukan *command move* atau *dig*, health power yang lebih besar dari nol, dan keadaan *current worm* atau *other worm* dalam bahaya atau tidak untuk menentukan *command* yang tepat. Fungsi objektif dari permainan ini adalah memenangkan permainan dengan cara mengeliminasi seluruh worms lawan, dengan menggunakan senjata atau skill yang tersedia dalam permainan ini.

3.2 Eksplorasi Alternatif Solusi Greedy

Berikut adalah beberapa hasil eksplorasi solusi greedy yang kelompok kami lakukan:

- Menargetkan musuh dengan jarak paling dekat dari cacing yang sedang aktif (*current worm*)
- Menargetkan musuh dengan HP paling sedikit dengan harapan dapat lebih cepat membunuh musuh
- Urutan prioritas senjata dari yang damagenya paling tinggi dengan harapan dapat lebih cepat membunuh musuh
- Penggunaan *banana bomb* ketika ada lebih dari 1 cacing lawan yang masuk dalam radius *banana bomb* ketika dilempar.(tidak di implementasi)
- Ketiga cacing si pemain berkumpul terlebih dahulu dengan jarak antar cacing lebih dari 2 satuan (karena tidak akan terkena efek *banana bomb* dan *snowball*) agar dapat menyerang musuh secara bersama-sama. (tidak di implementasi)
- Mengutamakan penggunaan *snowball* agar cacing lawan membeku kemudian berdasarkan strategi sebelumnya dimana ketiga cacing berkumpul maka si cacing lawan yang membeku dapat diserang bersamaan oleh ketiga cacing dan satu cacing bahkan dapat memakai *banana bomb*. (tidak di implementasi).

3.3 Analisis Efisiensi

Kompleksitas algoritma untuk strategi greedy pertama yaitu menargetkan cacing musuh dengan jarak yang terdekat dihitung dari berapa kali perhitungan jarak dilakukan dari cacing yang sedang aktif ke cacing musuh kemudian setelah didapat cacing yang terdekat maka dilanjutkan dengan mencari cell dengan jarak paling dekat dengan cacing musuh tersebut dalam 1 satuan. Perhitungan jarak dari cacing yang sedang aktif ke cacing musuh itu selalu dilakukan tiga kali sesuai dengan jumlah cacing musuh sehingga kompleksitasnya adalah O(1). Lalu, karena daerah permainannya merupakan sebuah matriks maka bisa disimpulkan pencarian cell terdekat paling dekat akan membuat si cacing mengecek sebanyak n x n cell di sekitarnya yang berarti kompleksitas algoritma untuk strategi ini adalah O(n^2). Algoritma ini kurang mangkus dalam beberapa situasi contohnya ialah saat cacing lawan terus bergerak menjauh.

Kompleksitas algoritma untuk strategi greedy kedua yaitu menargetkan cacing musuh dengan *health point* terendah dihitung juga berdasarkan jumlah pencarian cacing musuh dengan health power terendah dilanjut dengan pencarian cell terdekat dengan cacing tersebut. Maka strategi ini memiliki kompleksitas yang sama dengan strategi pertama yaitu O(n^2). Sama seperti strategi sebelumnya algoritma ini kurang mangkus dalam beberapa situasi seperti saat cacing pemain sangat jauh jaraknya dari cacing lawan.

Kompleksitas algoritma untuk strategi greedy ketiga yaitu urutan prioritas senjata dari yang damagenya paling tinggi dihitung dari operasi perbandingan apakah senjata dengan damage tertinggi dapat digunakan (terdapat musuh dalam radius serangan senjata tersebut, jumlah penggunaannya tidak melebihi batas). Kompleksitas dari operasi perbandingannya apakah senjata tersebut dapat digunakan adalah O(1) namun operasi pengecekan apakah target berada dalam radius senjata adalah O(n^2) karena harus melakukan pencarian pada daerah n x n apakah target ada dalam radius tersebut. Algoritma ini dapat lebih efektif jika lawan tidak menghindar ketika kita melakukan command menembak.

3.4 Analisis Efektivitas

1. Strategi greedy pertama yaitu menargetkan cacing musuh dengan jarak yang terdekat memberikan hasil yang optimal dengan dipengaruhi oleh banyak aspek lain seperti HP cacing pemain dan lawan, senjata yang dimiliki kedua cacing, dan lingkungan tempat keduanya berdekatan. Strategi ini juga memberikan cacing tujuan untuk awal yang pada awalnya si cacing tidak memiliki tujuan tertentu
2. Strategi greedy kedua yaitu menargetkan cacing musuh dengan *health point* dapat memberikan hasil yang optimal dikarenakan pemain dapat lebih unggul jika cacing yang dimiliki pemain memiliki HP yang lebih banyak daripada cacing lawannya.
3. Strategi greedy ketiga yaitu menentukan prioritas senjata dari yang damagenya paling tinggi akan memberikan hasil yang optimal dikarenakan cacing pemain dapat lebih banyak memberikan damage kepada cacing lawan yang kemungkinan besar menggunakan senjata secara acak.

3.5 Strategi Greedy yang Dipilih

Berdasarkan eksplorasi strategi greedy kemudian analisis efisiensi dan analisis efektivitas, kelompok kami memilih tiga strategi yang akan kami implementasikan dalam program kami yaitu strategi menargetkan cacing lawan dengan jarak terdekat, menargetkan cacing lawan dengan HP terendah, serta menentukan prioritas senjata dengan damage terbesar untuk digunakan. Beberapa alasan yang membuat kami memakai ketiga strategi tersebut adalah karena ketiga strategi tersebut sudah cukup optimal dan cukup mangkus serta mudah diimplementasikan oleh kelompok kami. Strategi yang lain mungkin lebih mangkus dan lebih

optimal akan tetapi kelompok kami kesulitan dalam mengimplementasikan strategi-strategi tersebut sehingga strategi greedy yang akan kami implementasikan di program kami adalah ketiga strategi diatas.

BAB 4
Implementasi dan Pengujian

4.1 Implementasi Program dalam Game Engine

```
public Command run()

    List<Cell> surroundingBlocks = getSurroundingCells(currentWorm.position.x,
    currentWorm.position.y)

    if (gameState.currentRound <= 6) then
    { selama 6 round cacing bergerak menuju tengah (setiap cacing 2 command pertamanya
    bergerak menuju tengah)}
        Cell block2 = goToMid(surroundingBlocks)
        if (block2.type == CellType.AIR) then
            return new MoveCommand(block2.x, block2.y)
        else if (block2.type == CellType.DIRT) then
            return new DigCommand(block2.x, block2.y);


    Worm target = getClosestEnemyWorm();
    {Mencari target yaitu antara cacing dengan jarak terdekat atau cacing dengan HP
    terkecil}
    if (getOpponentHP()<330) then
        target = getLeastHPEnemy()

    if (target.health>0) then
    { Cek jika HP target lebih lebih dari 0, Serang target dengan
    prioritas senjata yang damagenya paling tinggi (jika memenuhi syarat)}
        if (currentWorm.id == 2) then
            if (isOpponentInBananaRange(currentWorm, target)) then
                return new BananaShotCommand(target.position.x,
                target.position.y)


        if (currentWorm.id == 3) then
            if (isOpponentInSnowballRange(currentWorm, target)) then
                return new SnowBallCommand(target.position.x,
                target.position.y)
```

```
        {Jika tidak bisa menembak dengan senjata yang damagenya tinggi maka gunakan
         senjata basic}
        if (isOpponentInShootRange(currentWorm, target)) then
                Direction direction = resolveDirection(currentWorm.position,
                target.position)
                return new ShootCommand(direction)


{Mengecek jika ada teman yang sedang dalam bahaya}
boolean friendInDanger = isOtherWormInDanger()
if (gameState.myPlayer.remainingWormSelections>0) then
        if (friendInDanger) then
                MyWorm friendDanger = getFriendInDanger()
                Worm enemyDanger = getEnemyFriendInDanger()
                String s_command = ""
                if (friendDanger!=null && enemyDanger!=null) then
                        if (friendDanger.id == 2) then
                                if (isOpponentInBananaRange(friendDanger,
                                enemyDanger)) then
                                        s_command = String.format("banana %d %d",
                                        enemyDanger.position.x, enemyDanger.position.y)


                        if (friendDanger.id == 3) then
                                if (isOpponentInSnowballRange(friendDanger,
                                enemyDanger)) then
                                        s_command = String.format("snowball %d %d",
                                        enemyDanger.position.x, enemyDanger.position.y)


                        if (!s_command.equals("")) then
                                return new SelectCommand(friendDanger.id, s_command)



{Cek jika cacing saat ini dalam bahaya dan dapatkan musuh yang akan menyerangnya}
Worm enemyCurrDanger = getCurrWormInDanger()
if (enemyCurrDanger != null) then
        if (enemyCurrDanger.health > 0) then
                if (currentWorm.id == 2) then
                        if (isOpponentInBananaRange(currentWorm, enemyCurrDanger))
                        then
                                return new
```

```
                    BananaShotCommand(enemyCurrDanger.position.x,
                    enemyCurrDanger.position.y)


            if (currentWorm.id == 3) then
                if (isOpponentInSnowballRange(currentWorm,
                enemyCurrDanger)) then
                    return new
                    SnowBallCommand(enemyCurrDanger.position.x,
                    enemyCurrDanger.position.y)


            if (isOpponentInShootRange(currentWorm, enemyCurrDanger)) then
                Direction direction = resolveDirection(currentWorm.position,
                enemyCurrDanger.position)
                return new ShootCommand(direction)



{Strategi greedy bergerak ke cell menuju target dengan jarak atau HP paling sedikit}
Cell block = goToEnemyCell(surroundingBlocks, target)

if (friendInDanger) then
    Worm newTarget = getEnemyFriendInDanger()
    if(newTarget!=null) then
        block = goToEnemyCell(surroundingBlocks, newTarget)


if (block.type == CellType.AIR) then
    return new MoveCommand(block.x, block.y)
else if (block.type == CellType.DIRT) then
    return new DigCommand(block.x, block.y)


{Jika tidak ada yang memenuhi maka cacing akan bergerak secara random}
int cellIdx = random.nextInt(surroundingBlocks.size())
Cell block1 = surroundingBlocks.get(cellIdx)
if (block1.type == CellType.AIR) then
    return new MoveCommand(block1.x, block1.y)
else if (block1.type == CellType.DIRT) then
    return new DigCommand(block1.x, block1.y)


return new DoNothingCommand()
```

4.2 Penjelasan Struktur Data

Dalam permainan Worms ini informasi mengenai game state pada setiap round nya terdapat pada file state.json. Kami kemudian menyimpan informasi tersebut ke dalam class pada bahasa pemrograman java. Berikut adalah class yang kami buat

1. Class GameState
   Class ini memiliki atribut currentRound, maxRound, mapSize, currentWormId, consecutiveDoNothingCount, myPlayer, opponent, map. Class ini digunakan untuk menyimpan state game secara general.
2. Class MyPlayer
   Class ini memiliki atribut id, score, health, worms, remainingWormSelection. Class ini digunakan untuk menyimpan informasi umum tentang permainan kami.
3. Class Worm
   Class ini memiliki atribut id, health, position, diggingRange, movementRange, bananaBomb, snowballs. Class ini digunakan untuk menyimpan data worm.
4. Class MyWorm
   Class ini merupakan extend dari class worm dengan atribut tambahan weapon.
5. Class Weapon
   Class ini memiliki atribut damage dan range dari basic weapon.
6. Class Opponent
   Class ini memiliki atribut id, score, dan worms. Class ini menyimpan informasi mengenai musuh.
7. Class Position
   Class ini memiliki atribut x dan y. Class ini digunakan untuk menyimpan informasi mengenai posisi (x, y).
8. Class Cell
   Class ini memiliki atribut x, y, type, dan power up. Class ini digunakan untuk menyimpan informasi mengenai cell penyusun map.
9. Class PowerUp
   Class ini memiliki atribut type dan value. Class ini digunakan untuk menyimpan informasi mengenai power up pada suatu cell.
10. Class SnowBall
    Class ini memiliki atribut freezeDuration, range, count, dan freezeDuration. Class ini digunakan untuk menyimpan informasi mengenai senjata snowball.
11. Class BananaBomb
    Class ini memiliki atribut damage, range, count, damage radius. Class ini digunakan untuk menyimpan informasi mengenai senjata banana bomb.

12. Class BananaShotCommand
    Class ini digunakan jika kita memilih untuk melakukan command banana bomb.
13. Class Command
14. Class DigCommand
    Class ini digunakan jika kita memilih untuk melakukan command dig dengan parameter x dan y.
15. Class DoNothingCommand
    Class ini digunakan jika kita memilih untuk command do nothing
16. Class MoveCommand
    Class ini digunakan jika kita ingin memilih move command dengan parameter x dan y
17. Class SelectCommand
    Class ini digunakan ika kita ingin memilih select command dengan parameter id (id worm mana yang ingin diselect) dan command apa yang ingin dilakukan pada worm tersebut.
18. Class ShootCommand
    Class ini digunakan jika kita ingin memilih shoot command dengan parameter berupa direction
19. Class SnowBallCommand
    Class ini digunakan jika kita ingin memilih snowball command dengan parameter x dan y
20. Class CellType
    Class ini menyimpan tipe dari cell yaitu AIR, DIRT, DEEP SPACE.
21. Class Direction
    Tipe bentukan berupa arah seperti pada arah mata angin dan berguna sebagai arah untuk melakukan shot command.
22. Class PowerUpType
    Class ini berisi type power up yaitu HEALTH_PACK

4.3 Analisis dari Desain Solusi Algoritma *Greedy*

Strategi yang kami usulkan merupakan strategi yang akan terus mencoba menyerang musuh. Setiap round akan terus mendekati target (memiliki HP paling rendah atau jarak paling dekat) dan jika target dalam jangkauan serangan maka akan langsung diserang. Strategi menargetkan musuh dengan jarak terdekat atau HP terendah ini diharapkan dapat dengan cepat mendekati musuh dan langsung bisa menyerangnya. Namun, pada saat dilakukan pengujian, ketika kami bergerak mendekati target dan kemudian di ronde selanjutnya target tersebut aktif dan menyerang cacing kami tersebut maka hal tersebut dapat menyebabkan health point tim kami tertinggal dengan musuh dan dapat menyebabkan kekalahan. Sehingga strategi tersebut akan optimal jika cacing tim kami yang menyerang musuh terlebih dahulu. Lalu strategi pemilihan senjata dapat optimal jika lawan tidak menghindar dengan melakukan command move karena

jika lawan menghindar, maka akan menyebabkan damage yang diterima lawan lebih sedikit dan hal tersebut menjadi tidak optimal. Lalu, strategi yang kami buat ini belum memperhatikan health pack karena cacing kami akan bergerak mendekati target sehingga jarang bahkan tidak pernah mengambil health pack tersebut (meskipun sudah ada penanganan jika berada disekitar cell yang memiliki health pack tersebut maka akan diambil) dan kemungkinan besar akan diambil lawan sehingga HP lawan akan lebih unggul dan menyebabkan tim kami tertinggal dari lawan.

BAB 5
Kesimpulan dan Saran

5.1 Kesimpulan

Strategi yang kami buat dalam permainan ini memang belum sempurna tapi setidaknya kami telah mencoba untuk menerapkan strategi greedy dalam permainan ini dengan harapan langkah yang diambil setiap roundnya dapat membuat kami memenangkan permainan ini. Strategi greedy yang kami terapkan adalah menargetkan musuh dengan health point terendah atau jarak terdekat dan strategi pemilihan senjata dengan urutan prioritas dari yang damagenya paling besar.

5.2 Saran

Dalam pengerjaan tugas besar ini memang terkendala di bagaimana cara mengedit botnya dan mekanisme program utama permainan ini sehingga menyebabkan pengerjaan tugas besar ini dilakukan mendekati tenggat waktu. Tapi ternyata setelah dilakukan eksplorasi pada akhirnya mengerti bagaimana cara mengerjakan tugas besar ini. Saran agar kedepannya lebih baik lagi adalah coba eksplore dulu pada awal rilis tubes sehingga tidak bingung saat mendekati tenggat waktu.

Dalam pengerjaan tugas ini juga kami terdapat kendala komunikasi sehingga panik ketika mendekati deadline dan terjadi ketidakmerataan pembagian tugas. Saran agar kedepannya lebih baik adalah ketika mendekati tenggat waktu, komunikasi usahakan harus lancar.

Daftar Pustaka

- Munir, Rinaldi. 2021. Diktat Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika STEI ITB.
- http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Tugas-Besar-1-IF2211-Strategi-Algoritma-2021.pdf (diakses 15 Februari 2021 pukul 20.00)
- https://github.com/EntelectChallenge/2019-Worms/blob/develop/game-engine/game-rules.md (diakses 14 Februari 2021 pukul 21.00)