LAPORAN TUGAS BESAR I

IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma *Greedy* dalam pembuatan *bot* permainan Diamonds

Disusun oleh:

Benjamin Sihombing	(13522054)
Suthasoma Mahardhika Munthe	(13522098)
Marvin Scifo Y. Hutahaean	(13522110)

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Diamonds merupakan suatu *programming challenge* yang mempertandingkan *bot* yang anda buat dengan *bot* dari para pemain lainnya. Setiap pemain akan memiliki sebuah *bot* dimana tujuan dari *bot* ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing *bot*-nya.

Pada tugas pertama Strategi Algoritma ini, akan digunakan sebuah *bot* yang nantinya akan dipertandingkan satu sama lain. Tentunya pemain harus menggunakan strategi *greedy* dalam membuat *bot* ini. Terdapat dua komponen dari permainan Diamonds ini yaitu *game engine* dan *bot starter pack*.

Diamonds *game engine* secara umum berisi kode *backend* permainan yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *frontend* dan program *bot* dan kode *frontend* permainan yang berfungsi untuk memvisualisasikan permainan. Adapun tautan untuk melihat *game engine* adalah sebagai berikut:
https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0

Diamonds *bot starter pack* secara umum berisi program untuk memanggil API yang tersedia pada *backend*, program *bot logic* (bagian ini yang akan diimplementasikan dengan algoritma *greedy* pada *bot*), dan program utama dengan utilitas lainnya. Adapun tautan untuk melihat *bot starter pack* adalah sebagai berikut:
https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1

Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds

   Untuk memenangkan pertandingan, pemain harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru

bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

2. Red Button/Diamond Button

   Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

3. Teleporters

   Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika *bot* melewati sebuah teleporter maka *bot* akan berpindah menuju posisi teleporter yang lain.

4. Bots and Bases

   Pada game ini kita akan menggerakkan *bot* untuk mendapatkan diamond sebanyak banyaknya. Semua *bot* memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score *bot* akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) *bot* menjadi kosong.

5. Inventory

   Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, *bot* bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.

Cara kerja permainan Diamonds adalah sebagai berikut.

1. Pertama, setiap pemain (*bot*) akan ditempatkan pada board secara random. Masing-masing *bot* akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap *bot* diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama *bot* adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.

4. Setiap *bot* juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu *bot* harus segera kembali ke home base.
5. Apabila *bot* menuju ke posisi home base, score *bot* akan bertambah senilai diamond yang tersimpan pada inventory dan inventory *bot* akan menjadi kosong kembali.
6. Usahakan agar *bot* anda tidak bertemu dengan *bot* lawan. Jika *bot* A menimpa posisi *bot* B, *bot* B akan dikirim ke home base dan semua diamond pada inventory *bot* B akan hilang, diambil masuk ke inventory *bot* A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh *bot* telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

BAB II

LANDASAN TEORI

A. Algoritma *Greedy*

Algoritma *Greedy* adalah algoritma yang sering digunakan untuk menyelesaikan persoalan optimasi. *Greedy* sendiri bisa diartikan rakus atau tamak. Algoritma ini dinamakan *greedy* karena algoritma ini berkelakuan seperti orang yang rakus atau tamak. Algortima ini mengambil langkah/keputusan yang cepat, namun langkah/keputusan yang diambil belum tentu paling optimal. Hal ini disebabkan oleh proses di dalam algoritmanya yang membagi permasalahan menjadi bagian-bagian. Algoritma akan mengambil langkah/keputusan yang paling optimal pada bagian permasalahan yang sedang ditangani saja tanpa melihat bagian permasalahan kedepannya. Padahal, mungkin saja keputusan yang diambil merugikan dalam bagian permasalahan kedepannya. Akibatnya, langkah-langkah yang diambil tersebut bukanlah solusi yang paling optimal untuk keseluruhan masalah. Di sisi lain, algoritma *greedy* memiliki kelebihan di kecepatan pemrosesannya karena algortima ini tidak mengecek semua solusi yang memungkinkan.

B. Cara Kerja Program

Untuk menjalankan *bot*, caranya masih sama dengan cara menjalankan *bot starter pack* jadi tidak ada bagian menjalankan program yang diubah oleh kelompok kami. *Bot* melakukan perjalanannya dengan berjalan 1 arah ke kiri, kanan, atas, atau bawah dan untuk jalan *bot* memerlukan waktu 1 detik ditambah dengan *runtime* dari program yang dibuat. Terkait dengan kemana *bot* tersebut berjalan diatur dengan algoritma yang didasarkan oleh strategi yang dipikirkan oleh anggota kelompok. Pada kasus ini, strategi yang digunakan adalah strategi *greedy*.

Program dalam bentuk awalnya memiliki strategi untuk memilih arah secara acak. Algoritma yang mengatur jalannya *bot* tersebut ada pada *directory* ‘tubes1-IF2211-bot-starter-pack-1.0.1’. Pada *directory* tersebut terdapat file ‘game’ dan pada file ‘game’ terdapat file yang bernama ‘logic’ yang berisi sebuah file yang bernama ‘random.py’. Program yang akan diganti ada pada bagian class ‘RandomLogic’. Di class tersebut dan di fungsi

yang bernama NextMove, kelompok kami mengimplementasikan strategi *greedy* tersebut untuk membuat *bot* yang paling optimal dalam mencari *diamond*.

(Penjelasan singkat terkait strategi *bot* berjalan - Nanti, suta belom jadi *botnya*)

Game engine yang digunakan dalam permainan ini sudah disediakan oleh sebuah folder yang bernama 'tubes1-IF2211-game-engine-1.1.0'. Sebelum memainkan game ini, pemain perlu memasang ketiga aplikasi ini yaitu Node.js, Docker desktop, dan Yarn. Pada Docker, pemain juga harus membuat database yang diperlukan oleh aplikasi berjalan pada Docker tersebut. Setelah semuanya selesai, program perlu melakukan *build* terlebih dahulu sebelum dijalankan dengan masuk ke website "http://localhost:8082/ untuk memulai permainan tersebut. Beginilah bentuk dari program yang akan dimainkan.

Gambar 1.3.1 Tampilan Permainan

Untuk menjalankan *bot* pada halaman web ini, terdapat 2 cara yang bisa dilakukan. Namun sebelum itu, pastikan pemain sudah memasang *dependencies* yang dibutuhkan oleh program. Cara pertama untuk menggunakan *bot* adalah dengan menjalankan command ini (nama, email, password bisa diganti):

```
python main.py --logic Random --email=your_email@example.com --name=your_name
--password=your_password --team etimo
```

Terdapat cara lain yang bisa digunakan untuk memainkan Diamonds dengan beberapa *bot* sekaligus dan dengan strategi yang berbeda-beda. Beginilah caranya.

- Untuk Windows

```
./run-bots.bat
```

- Untuk Linux/ macOS (mungkin)

```
./run-bots.sh
```

Catatan: Untuk latihan atau dalam permainan tanpa regulasi, pemain bisa mengubah komponen dalam *game engine* tersebut. Contohnya adalah dengan mengganti jumlah *diamond* yang ada, ukuran *bot*, ukuran papan, dll.

BAB III

APLIKASI STRATEGI *GREEDY*

A. Elemen-elemen Algoritma *Greedy*

Pada algoritma *greedy* terdapat 6 elemen yang diperlukan untuk menjalankan algoritma. Berikut ini, elemen-elemen tersebut:

1. Himpunan Kandidat (C): berisi kandidat berupa list beberapa objek pada board, yaitu diamonds, red button, sepasang teleporters, dan base.
2. Himpunan Solusi (S): berisi kandidat atau objek yang sudah dipilih.
3. Fungsi Solusi: memeriksa dan memilih jalur yang berasal dari himpunan kandidat dan solusi yang dipilih *bot* untuk dilewati menuju obyek tujuan
4. Fungsi Seleksi: memilih objek diamond yang tersedia di board dengan value paling kecil
5. Fungsi Kelayakan: memeriksa apakah obyek diamond yang diambil masih layak untuk diambil (inventory masih cukup), jika sudah penuh (atau semi penuh) maka *bot* akan kembali ke base terlebih dahulu
6. Fungsi Objektif: memeriksa apakah tujuan masih dapat dicapai dengan sisa waktu yang ada, atau mengutamakan objek lain jika obyek tersebut berada pada area pergerakan *bot* menuju objek tujuan awal.

Dalam permainan Diamonds, beberapa persoalan bisa dibagi menjadi objek-objek yang ada di Diamonds. Untuk melakukan pembagian menjadi beberapa objek, *mapping* perlu dilakukan untuk mendapatkan hasil sebagai berikut:

Tabel 3.1.1 Komponen atau Objek di dalam Permainan

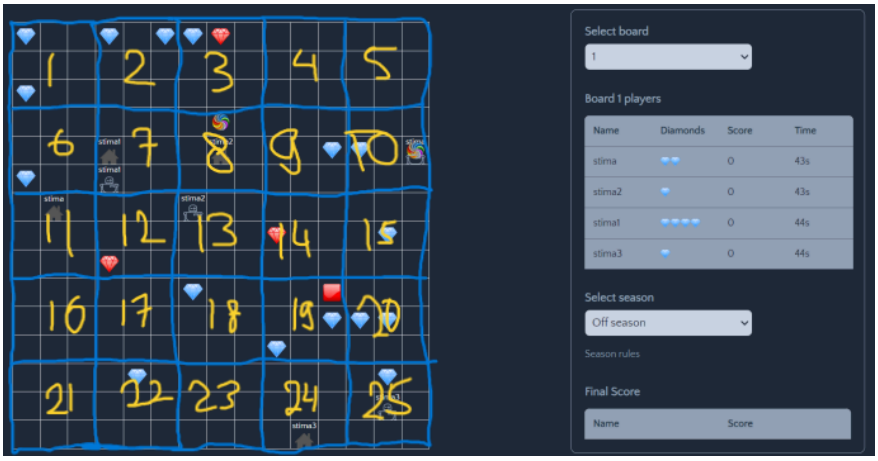
No	Persoalan	Kegunaan	Fungsi proses
1	*Diamond*	- Menambahkan 1 poin dari 5 poin ke dalam *inventory* dari *bot* - Dijangkau oleh *bot* yang menggunakan algoritma *greedy* - Hanya bisa didapat jika *diamond* di inventory kurang dari sama dengan 5	*MachineBot* (Fungsi nya bernama *nextMove*)
2	*Red Diamond*	- Menambahkan 2 poin dari 5 poin ke dalam *inventory* dari *bot*	*MachineBot* (Fungsi nya bernama *nextMove*)

		- Dijangkau oleh bot menggunakan algoritma *greedy* - Hanya bisa didapat jika *diamond* di inventory kurang dari sama dengan 4	
3	*Diamond Button*	- Me-*reset* susunan objek yang ada di dalam game - Dijangkau oleh *bot* yang menggunakan algoritma *greedy*	*MachineBot* (Fungsi nya bernama *nextMove*)
4	*Bot*	- Komponen yang digunakan oleh pengembang untuk mengaplikasikan algoritma *greedy* pada komponen tersebut	*MachineBot* (Fungsi nya bernama *nextMove*)
5	*Teleporter*	- Mengganti posisi *bot* dari teleporter A ke teleporter B	*MachineBot* (Fungsi nya bernama *nextMove*)
6	*Base*	- Komponen yang digunakan bagi pemain untuk menyimpan semua berlian mereka	*MachineBot* (Fungsi nya bernama *nextMove*)
7	*Inventory*	- Jumlah *diamond* yang ada di dalam *inventory*	*MachineBot* (Fungsi nya bernama *nextMove*)

B. Alternatif-alternatif Solusi *Greedy*

Terdapat beberapa alternatif solusi yang bisa dilakukan untuk mendapatkan *diamond* yang banyak dengan menggunakan algoritma *greedy*. Berikut adalah alternatif-alternatif yang kami sajikan.

1. *Greedy* by area
   Strategi ini melibatkan pembagian papan permainan menjadi beberapa sektor. Kompleksitas dari algoritma ini adalah O(n). Misalkan papan permainan berukuran 15x15, maka papan tersebut akan dibagi menjadi 25 sektor.

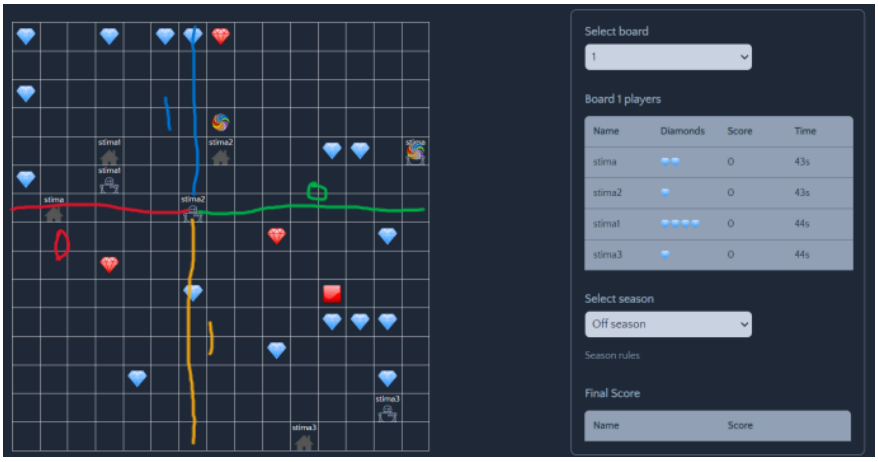


Gambar 3.2.1 Ilustrasi Algortima *Greedy by* Area

Misalkan pemain adalah ‘stima2’. Pemain ‘stima2’ ini akan mencari zona yang paling dekat dengan dia dan jumlah diamond yang ada pada zona tersebut. Jumlah diamond yang di zona tersebut akan dibagi dengan jarak zona ke jarak zona dimana ‘stima2’ berada. Hal ini dilakukan sampai waktu habis.

2. *Greedy* by direction density

   Strategi ini melibatkan pembagian papan permainan menjadi beberapa arah. Kompleksitas dari algoritma ini adalah O(n). Misalkan papan permainan berukuran 15x15, maka papan tersebut akan dibagi menjadi 4 arah.

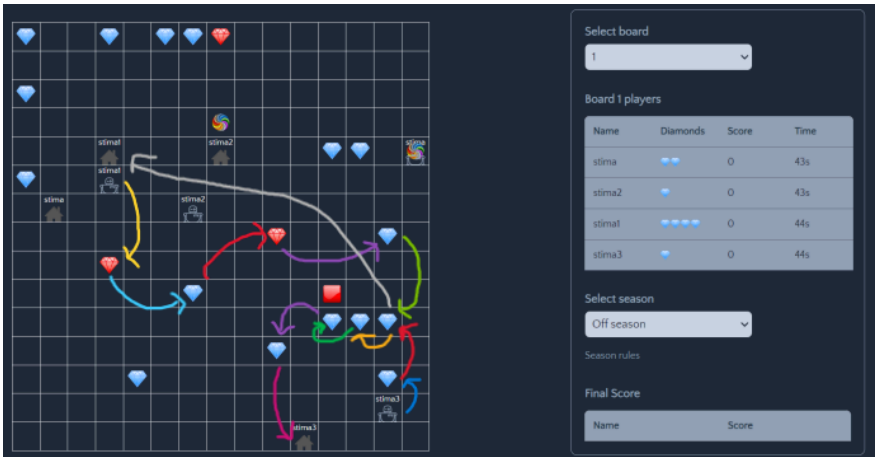


Gambar 3.2.2 - Ilustrasi Algoritma *Greedy by direction density*

Misalkan pemain adalah ‘stima2’. Pemain ‘stima2’ ini akan mencari arah yang memiliki jumlah diamond yang paling banyak. Sebenarnya *bot* tersebut tidak akan pergi menggapai semua *diamond* tersebut tetapi hanya pergi ke arah *diamond* tersebut berada sampai menemukan arah yang memiliki lebih banyak *diamond*. Proses ini dilakukan sampai waktunya habis.

3. *Greedy* by steps

Strategi ini melibatkan pencarian pijakan yang dibutuhkan oleh *bot* dan semua *diamond* yang ada di papan. Kompleksitas dari algoritma ini adalah O(n). Contoh simulasi dari *greedy* by steps dapat dilihat di gambar berikut ini.



Gambar 3.2.3 - Ilustrasi Algoritma *Greedy by steps*

Selain *diamond*, pemain juga memprioritaskan pergi ke tombol merah jika memang jaraknya yang paling dekat (Ini dilakukan agar *diamond*-nya muncul lagi dengan jumlah yang lebih banyak). Misalkan pemain adalah 'stima3'. Pemain 'stima3' akan mencari semua kemungkinan pijakan yang dibutuhkan untuk mencapai *diamond* atau *diamondbutton* tersebut. Pemain 'stima3' akan menggapai *diamond* tersebut dengan pijakan seminimal mungkin. Dan setelah itu, pemain akan mencari *diamond* terdekat lainnya. Proses ini dilakukan sampai waktunya habis.

4. Combination of *greedy* strategies with formula

   Algoritma ini mengukur jarak *bot*-diamond, diamond ke diamond lain, dan jarak diamond ke base jadi untuk mengukur jarak diamond ke diamond lain mempunyai kompleksitasnya O(n^2) untuk menentukan diamond tujuan, kami memakai rumus, misalnya:

$$d = jarak\ bot\ ke\ diamond$$
$$o = jarak\ diamond\ ke\ diamond\ lain$$
$$b = jarak\ bot\ ke\ base$$
$$p = point\ diamond$$
$$value = \frac{d^2 ob}{p+3}$$

   Apapun yang punya value paling kecil dari setiap diamond akan menjadi tujuan utama dari *bot* jika waktunya habis dengan prioritasnya untuk kembali ke base.

Selain strategi *greedy*, *bot* juga bisa melakukan hal-hal lainnya untuk memaksimalkan jumlah *diamond* yang dimilikinya. Contohnya adalah sebagai berikut.

1. *Find the most diamond*
   Diamond yang berwarna merah memiliki 2 poin sedangkan yang biru memiliki 1 poin. Misalkan terdapat kasus perbandingan 1 hasil *greedy* dengan 1 hasil *greedy* yang lain adalah sama. *Bot* akan memprioritaskan pergi ke *diamond* yang memiliki 2 poin.
2. *Go home when time is running out*
   Tidak ada gunanya *diamond* jika tidak dibawa pulang. Misalkan waktu tinggal sedikit lagi dan *bot* memiliki beberapa *diamond*. Maka *bot* akan dipaksa pulang agar perjuangan *bot* sebelumnya tidak sia-sia.
3. *Go home when you're close*
   Jika *bot* sedang berada pada posisi yang sangat dekat dengan *base* dan mempunyai sejumlah *diamond*, tidak ada salahnya bagi *bot* untuk menaruh *diamond*-nya saja dan lanjut mencari *diamond* yang lain. Hal ini dilakukan tidak hanya untuk memastikan penambahan poin tetapi agar *bot* bisa menampung lebih banyak *diamond* dalam perjalanan.

C. Analisis Efisiensi dan Efektivitas Algoritma *Greedy*

Dari hasil *testing* yang kami lakukan terhadap beberapa alternatif solusi yang kami lakukan. Beginilah hasil dari *testing* ini. Poin yang terkecil akan menjadi algoritma *greedy* yang terbaik.

Note: Jika mempunyai poin yang sama (Misalkan peringkat 1 dan 2 punya poin yang sama), kedua *bot* akan mendapatkan 1,5 poin (Dan seterusnya untuk peringkat lainnya).

Tabel 3.3.1 Hasil Pertandingan Antar *Bot*

No	Greedy by Area	Greedy by Direction	Greedy by Steps	Greedy Combinations
1	3.5	3.5	1	2
2	3	2	4	1
3	3	4	1	2
4	2	3	1	4
5	4	3	2	1
6	3	4	1	2
7	4	3	1	2
8	1	4	3	2

9	2.5	4	2.5	1
10	1.5	3	1.5	4
11	2	3	1	4
12	1	4	2	3
13	2	4	3	1
14	1	3	2	4
15	3	4	2	1
16	3	4	2	1
17	3.5	2	3.5	1
18	4	3	1	2
19	4	2.5	2.5	1
20	3	4	2	1
21	1.5	4	3	1.5
22	4	3	2	1
23	1	2	3	4
24	3	4	1.5	1.5
25	2.5	4	2.5	1
Total	66	84	51	49

Tabel 3.3.1 - Data hasil *testing* dari beberapa algoritma *greedy*

Berdasarkan data yang diberikan, *greedy* dengan kombinasi berbagai strategi *greedy* mempunyai efisiensi yang besar. Strategi *greedy* yang lain mungkin juga bisa menyelesaikan pekerjaan dalam menyelesaikan permainan Diamond tetapi strategi tersebut tidak cukup cepat dalam mencari semua *diamond* yang ada. Berikut adalah penjelasannya.

1. *Greedy* by Area kurang baik untuk digunakan karena pencariannya dengan menggunakan blok saja (blok yang digunakan 3x3) sehingga mempunyai resiko bagi *bot* untuk menempuh jarak yang lebih panjang meskipun pada blok tersebut terdapat banyak *diamond* padahal sepanjang jalan untuk mencapai *diamond* tersebut *bot* seharusnya mengambil *diamond* tersebut terlebih dahulu.

2. *Greedy* by Direction kurang baik untuk digunakan karena pencariannya hanya berdasarkan *line of sight* dari *bot* tersebut dari segala arah sehingga *diamond* yang tidak berada pada hadapan *bot* tidak akan dihiraukan meskipun terdapat kemungkinan bahwa banyak *diamond* yang terdapat di sekeliling *bot* tersebut. *Bot* kemungkinan bisa mencapai *diamond* tersebut tetapi juga mempunyai kemungkinan untuk menempuh jarak yang lebih jauh.
3. *Greedy* by Steps meskipun mempunyai nilai yang dekat dengan *greedy combination*, strategi tersebut masih mempunyai kelemahan. Tidak seperti *combination*, *Greedy* by Steps tidak melakukan pemeriksaan terhadap jarak ke *base* dan kerapatan *diamond* yang ada di daerahnya. Strategi ini hanya berfokus terhadap diamond mana yang paling dekat dengan *bot* tersebut sehingga mempunyai resiko untuk pergi ke *diamond* di daerah yang sepi *diamond*-nya meskipun beberapa *diamond* yang jauh berada di tempat yang kerapatan *diamond*-nya besar.

D. Strategi *Greedy* Terpilih

Setelah melakukan beberapa *testing*, kami memutuskan bahwa strategi yang paling cocok untuk memainkan permainan Diamonds adalah *greedy* by *combination*. *Greedy by combination* ini mencampurkan beberapa strategi *greedy* menjadi 1 dalam bentuk sebuah rumus. Strategi *greedy* yang dicampurkan adalah *greedy by diamond to bot*, *greedy by diamond to diamond*, *greedy by nearest diamond to base*, dan faktor lainnya seperti jumlah *diamond* yang dimiliki *bot*. Dari semua strategi dan faktor yang diberikan. Sebuah rumus dibuat untuk menghasilkan strategi *greedy by combination* tersebut. Beginilah rumus untuk membuat strategi tersebut.

$$d = jarak\ bot\ ke\ diamond$$
$$o = jarak\ diamond\ ke\ diamond\ lain$$
$$b = jarak\ bot\ ke\ base$$
$$p = point\ diamond$$
$$value = \frac{d^2 ob}{p+3}$$

Perlu diketahui bahwa selain strategi tersebut, *bot* juga akan melakukan beberapa optimisasi untuk menambah kemungkinan untuk mendapatkan banyak poin. Contohnya adalah pulang ke *base* jika waktu sudah mau habis, singgah ke *diamond* yang berada pada area pergerakan *bot* saat menuju *base*, saat *bot* menuju *diamond* dengan *base* pada area pergerakannya, *bot* akan singgah pada base terlebih dahulu, dan beberapa kejadian lainnya yang akan dijelaskan pada Bab IV.

Greedy by combination tidak hanya memiliki banyak keuntungan yang merupakan hasil dari kombinasi beberapa strategi *greedy* tetapi juga mempunyai hasil data sebagai bukti dari keuntungan tersebut. Terlihat dari data pada bagian C, *Greedy by*

combination memiliki poin yang terkecil jika dibandingkan dengan strategi-strategi *greedy* lainnya. Strategi *greedy* yang lain tersebut tidak sembarang dibuat dan pernah menjadi peringkat satu dari semua 25 permainan tersebut. Jadi, bukanlah tugas yang mudah bagi strategi *greedy* yang kami pakai untuk mengalahkan strategi mapan tersebut dan strategi ini terbukti bisa mengalahkan strategi lain yang kelompok kami sediakan. Oleh karena itu, pada laporan ini, kami menggunakan *greedy by combination*.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Algortima

Startegi greedy terpilih diimplementasikan seperti *pseudocode* di bawah ini. Proses seleksi *diamond* yang ada pada board dipilah terlebih dahulu setiap kali fungsi *next_move* dipanggil untuk mengevaluasi keadaan board secara *real-time*. Fungsi *pickDiamond* adalah fungsi seleksi yang akan melakukan evaluasi terhadap diamond yang sudah dipilah dengan fungsi *initializeData*. Evaluasi pada pickDiamond akan melakukan perhitungan nilai (*value*) untuk setiap diamond sesuai dengan formula yang telah dijelaskan di atas. Fungsi *pickDiamond* akan memilih *diamond* yang masih dapat dimuat ke dalam *inventory* dengan *value* yang paling kecil.

Value setiap *diamond* yang bernilai paling kecil memiliki beberapa sifat yaitu, paling dekat dengan *bot*, relatif dekat dengan *diamond* lain, dekat dengan *base*, dan poinnya relatif lebih besar. Artinya dengan *value* yang kecil, *diamond* tersebut berada pada area yang banyak muncul diamond. Algoritma ini tidak memperhitungkan keberadaan *bot* lain. Kami telah melakukan uji coba dan mendapatkan data bahwa setiap *bot* yang mempertimbangkan *bot* lain cenderung menghindari area yang jumlah diamondnya banyak karena *bot-bot* lain pada umumnya mengincar area tersebut.

Fungsi program utama: **function** next_move()

```
function next_move(board_bot: GameObject, board: Board) -> direction
{ fungsi utama yang akan mengembalikan arah pergerakan bot }
Deklarasi:
    priorityDiamond : Position
    diamondPick : Position

Algoritma:
    initializeData()     {melakukan inisialisasi semua field class}
    if not timeLeftStillEnough() then
        if not emptyInventory(bot) then
            if not isEqualPosition(bot.position, bot.base) then
                priorityDiamond <- pickDiamond()
                if priorityDiamond and isInAreaMove(bot.base, priorityDiamond) then
                    return getSaveDirection(priorityDiamond)
                if isInAreaMove(bot.base, redButton.position) then
```

```
                return getSaveDirection(redButton.position)
            return getSaveDirection(bot.base)
                                        { getSaveDirection(destination) }
                                        { mengembalikan arah yang aman}
                                        { dari teleport menuju destination }

        else
            return getRandomMove() { memberikan gerakan sembarang yang masih valid }
    else
        priorityDiamond <- pickDiamond()
        if (priorityDiamond != None) and isInAreaMove(bot.base, priorityDiamond) then
            return getSaveDirection(priorityDiamond)

        else
            if isInAreaMove(bot.base, redButton.position) then
                return getSaveDirection(redButton.position)
            if  not isEqualPosition(bot.position, bot.base) then
                return getSaveDirection(bot.base)
            else
                return getRandomMove()
else
    if not fullInventory(bot) then
        priorityDiamond <- pickDiamond()
        if (priorityDiamond != None) then
            diamondPick <- priorityDiamond
            if self.isInAreaMove(diamondPick, bot.base) then
                return getSaveDirection(bot.base)
            if not emptyInventory(bot) and diamondTooFar() then
                return getSaveDirection(bot.base)
            return getSaveDirection(diamondPick)
        else
            if not isEqualPosition(bot.position, bot.base) then
                return getSaveDirection(bot.base)
            return getRandomMove()
    else
        return getSaveDirection(bot.base)
```

Fungsi tambahan: **function** pickDiamond()

```
function pickDiamond()-> Diamond
Deklarasi:
    evaluateD : List[EvaluatedDiamond]
    countDiamonds : integer
    valueOfi : real

Algoritma:
    evaluateD <- evaluateDiamond()      { melakukan evaluasi nilai-nilai setiap diamond }
    countDiamonds <- length(evaluateD)
    if not emptyDiamond() then
        getDiamond <- (evaluateD[0],float("inf"))
        i traversal [0..countDiamonds-1]
            valueOfi <- getDiamondPriorityValue(evaluateD[i,0], evaluateD[i,1],
evaluateD[i,2], evaluateD[i,3])
            if lessValue(valueOfi) and inventoryFits(diamond) then
                getDiamond <- (evaluateD[i], valueOfi)
        if countDiamonds = 1 and not inventoryFits(diamond) then
            return None
        diamondPick <- getDiamond[0][0]
        return diamondPick
    return None
```

Fungsi tambahan: **function** evaluateDiamond()

```
function evaluateDiamond() -> List[Diamond]
Deklarasi:
    evaluation : List[EvaluatedDiamond]
    countDiamonds : integer

Algoritma:
    evaluation <- []
    countDiamonds <- len(listOfDiamonds)
    i traversal [0..countDiamonds-1]
        dist <- stepDistance(self.bot, listOfDiamonds[i])
        point <- listOfDiamonds[i].points
        temp_eval <- [listOfDiamonds[i].position,dist,0,point]
        j traversal  [0..countDiamonds-1]
```

```
            temp_eval[2] <- temp_eval + stepDistance(listOfDiamonds[i],listOfDiamonds[j])
        evaluation.append(temp_eval)
    return evaluation
```

Fungsi tambahan: **function** getDiamondPriorityValue()

```
function getDiamondPriorityValue(posDiamond : Position, distance : integer,
relativeDistance : integer, point : integer) -> real
Deklarasi:
    posBase : Position
    value : real

Algoritma:
    posBase <- self.bot.properties.base
    value <- self.getDistance(posDiamond, posBase)
    value <- distance*distance * relativeDistance * value
    value <- value/(point+3)
    return value
```

B. Penjelasan Struktur Data

Data yang diterima *bot* kami berupa *board_bot* dan *board*. Data *board_bot* adalah *bot* itu sendiri. *Bot* ini memiliki beberapa informasi yang dimuat, di antaranya adalah posisi *base*, sisa waktu, jumlah *diamond* pada *inventory*, ukuran *inventory*, dan posisi *bot*. *Bot* menyimpan data lain, namun kami hanya perlu akses terhadap data yang sudah disebutkan barusan saja. Bagian data *board* kami hanya mengakses *field game_objects.*

Struktur data yang kami gunakan untuk menyimpan data yang digunakan pada kelas *MachineBot* yang diturunkan dari kelas *BasicLogic* adalah sebagai berikut.

Tabel 4.2.1 Struktur Data yang Digunakan *Bot*

Data	Deskripsi
bot	Menyimpan data *bot* saat ini, seperti posisi, jumlah *diamond*, dan sisa waktu.
redButton	Menyimpan data objek Red Button.
listOfDiamonds	Data berupa list yang berisi beberapa objek *diamond*

	yang ada pada board saat ini.
listOfTeleport	Data berupa list sepasang objek Teleport.
board_width, board_height	Menyimpan dimensi board.
matrix	Menyimpan posisi objek Teleport untuk memudahkan akses posisi untuk menentukan *direction* selanjutnya terhindar dari objek yang menghambat perpindahan *bot*.

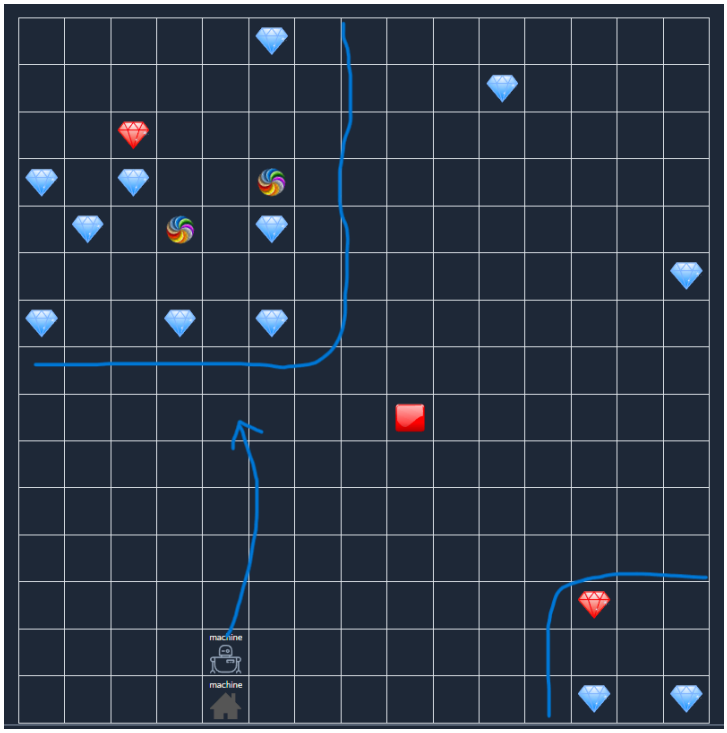
C. Analisis Desain Algoritma *Greedy*

Setelah dilakukan beberapa kali pengujian secara tunggal atau bertanding dengan *bot* lain, desain *bot* yang sudah dibuat ini lebih baik. *Bot* yang telah kami tentukan sebagai *bot* pilihan adalah *bot* yang memakai strategi *greedy* dengan kombinasi parameter berupa jarak *bot* ke *diamond*, jarak *diamond* ke *diamond* lain, poin, dan jarak *diamond* ke base. Sebelumnya kami memperhitungkan keberadaan *bot* lain sehingga *bot* kami akan memilih area yang memiliki cukup *diamond*, tetapi relatif jauh dari *bot* lawan. Namun, hasil sangat skor kurang memuaskan karena *bot* cenderung menghindari daerah dengan jumlah *diamond* yang relatif padat.

Jumlah skor yang diperoleh *bot* secara rata-rata berada di sekitar 15 poin, khususnya jika dijalankan secara tunggal pada satu *board*. Jika dilakukan pertandingan dengan beberapa *bot* alternatif lain, skor yang diperoleh beragam mulai dari 7 sampai 26 poin. Saat *base* dari *bot* berada jauh dari daerah yang pada *diamond* atau di-*tackle bot* lain, *bot* kehilangan poin dan tidak dapat mengumpulkan banyak *diamond* untuk memperoleh skor yang optimal. Sebaliknya, ada saat di mana *base bot* berada di pusat *board* atau daerah padat *diamond.* Pada saat tersebut *bot* dapat mengumpulkan banyak diamond, sehingga skor rata-ratanya adalah 15 poin. Pada kasus tertentu, *bot* dapat men-*tackle bot* lain sehingga rekornya *bot* buatan kami mencapai skor 26 poin.

Gambar 4.3.1 *Bot* kembali ke Base karena Sisa Waktu

Jika waktu sudah hampir habis (jarak *bot* ke *base* + 1 ≤ siswa waktu) *bot* akan memilih untuk kembali ke *base* (lihat Gambar 4.3.1). Selain kembali ke *base*, jika pada area gerak *bot* ke *base* terdapat *diamond* atau Red Button, maka *bot* akan singgah terlebih dahulu pada objek tersebut.



Gambar 4.3.2 *Bot* Menuju Daerah Padat *Diamond*

Bot akan lebih cenderung bergerak ke daerah yang padat *diamond* jika jarak ke *bot* relatif tidak terlalu jauh berbeda. Posisi *base* yang berada di posisi yang terlalu jauh dari daerah padat *diamond* menjadi faktor penyebab ketidakoptimalan skor yang diperoleh *bot*. Oleh karena itu, *bot* beberapa kali tidak dapat mencapai skor di atas 10 poin (lihat Gambar 4.3.2).

BAB V

KESIMPULAN DAN SARAN

A. Kesimpulan

Greedy adalah salah satu strategi algoritma yang melibatkan prioritas terhadap suatu faktor yang paling maksimum atau minimum. Salah satu persoalan yang bisa diselesaikan oleh algoritma *Greedy* ini adalah bermain pencarian Diamonds. Algoritma *greedy* yang kelompok kami rancang yang berupa *greedy by combination of greedy strategies* dapat digunakan untuk mendapatkan jumlah diamond yang cukup banyak. Algoritma yang digunakan dioptimasi dengan mempertimbangkan beberapa hal seperti jumlah pijakan, kerapatan *Diamond*, jarak ke *base*, dan jarak ke *Diamond* lainnya. Namun, *greedy* seperti ini juga mempunyai banyak kekurangan sebagai sebuah strategi algoritma sehingga terdapat strategi lain yang bisa digunakan untuk memaksimalkan hasil poin yang didapat.

B. Saran

Saran untuk meningkatkan kinerja kelompok kami yaitu:

1. Lebih baik dalam memahami struktur data dari *bot* yang digunakan. Hal ini dilakukan agar penulisan program bisa lebih mudah untuk dilakukan dengan memanggil struktur data yang disediakan.
2. Lebih banyak komunikasi antar kelompok untuk mengetahui progress dari masing-masing *bot* yang dibuat oleh anggota kelompok. Hal ini bisa dilakukan agar setiap anggota kelompok memberi saran untuk bagaimana cara mengoptimasi *bot* yang sedang dibuat
3. Lebih banyak melakukan *testing* agar bisa mengidentifikasi *bot* yang lebih konsisten. Sebaiknya dilakukan *testing* antara *bot* anggota kelompok agar bisa lebih cepat dalam mengidentifikasi kelemahan dari *bot* yang dibuat.

LAMPIRAN

Tautan *repository* GitHub:

sotul04/Tubes1_Stima (github.com)

Tautan video:

https://youtu.be/dZBVpTAV4K8

DAFTAR PUSTAKA

https://docs.google.com/document/d/1L92Axb89yIkom0b24D350Z1QAr8rujvHof7-kXRAp7c/edit

https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0

https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tubes1-Stima-2024.pdf24.docx (itb.ac.id)