

LAPORAN TUGAS BESAR 1
IF2211 - STRATEGI ALGORITMA

Pemanfaatan Algoritma *Greedy* dalam Pembuatan *Bot* permainan Diamonds



Kelompok Tipis - Tipis

Disusun oleh :

Raffael Boymian Siahaan	13522046
Naufal Adnan	13522116
Berto Ricardo Togatorop	13522118

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

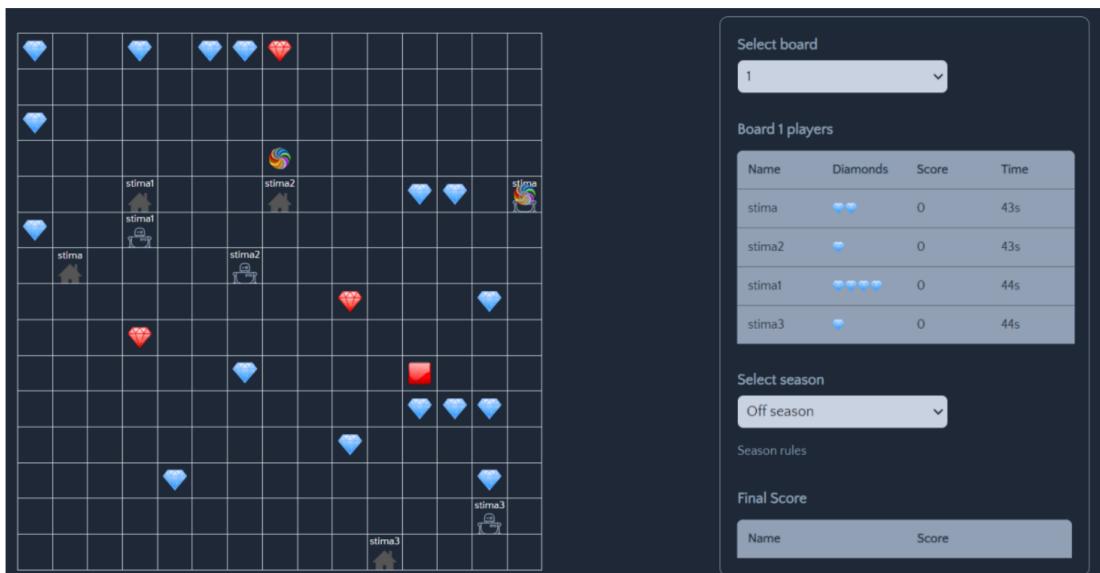
DAFTAR ISI

DESKRIPSI TUGAS.....	1
LANDASAN TEORI.....	5
2.1 Algoritma Greedy.....	5
2.1.1 Elemen Algoritma Greedy.....	5
2.1.2 Contoh-Contoh Persoalan yang Diselesaikan dengan Algoritma Greedy.....	7
2.2 Cara Kerja Program Secara Umum.....	7
APLIKASI STRATEGI GREEDY.....	11
3.1 Mapping Diamonds secara Umum.....	11
3.2 Alternatif utama Algoritma Greedy.....	11
3.2.1 Algoritma greedy berdasarkan jarak diamond terdekat.....	11
3.2.2 Algoritma greedy berdasarkan bobot diamond terbesar.....	13
3.2.3 Algoritma greedy berdasarkan bobot per move terbesar.....	14
3.2.4 Algoritma greedy berdasarkan jarak diamond dengan base.....	15
3.2.5 Algoritma greedy berdasarkan konsentrasi diamond dalam suatu daerah.....	17
3.2.6 Algoritma greedy berdasarkan inventory lawan.....	18
3.3 Strategi Greedy yang diimplementasikan.....	19
IMPLEMENTASI DAN PENGUJIAN.....	22
4.1 Implementasi dengan pseudocode.....	22
4.2 Struktur Data Program.....	28
4.3 Analisis dan Pengujian.....	30
KESIMPULAN.....	38
5.1 Kesimpulan.....	38
5.2 Saran.....	38
LAMPIRAN.....	40
DAFTAR PUSTAKA.....	40

BAB 1

DESKRIPSI TUGAS

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.



Gambar 1.1 Permainan Diamonds

Program permainan *Diamonds* terdiri atas:

1. *Game engine*, yang secara umum berisi:
 - a. Kode *backend* permainan, yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *frontend* dan program bot
 - b. Kode *frontend* permainan, yang berfungsi untuk memvisualisasikan permainan
2. *Bot starter pack*, yang secara umum berisi:
 - a. Program untuk memanggil API yang tersedia pada backend
 - b. Program bot *logic* (bagian ini yang akan kalian implementasikan dengan algoritma *greedy* untuk bot kelompok kalian)
 - c. Program utama (*main*) dan utilitas lainnya

Game engine dan bot dari *bot starter pack* diimplementasikan dari sumber yang telah tersedia pada pranala berikut.

- ***Game engine* :**

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

- ***Bot starter pack* :**

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

Komponen-komponen dari permainan Diamonds antara lain:

1. **Diamonds**



Gambar 1.2 Diamond Biru dan Merah

Untuk memenangkan pertandingan, kita 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**



Gambar 1.3 Red 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.

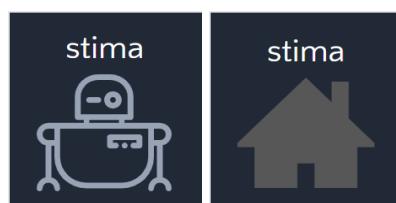
3. Teleporters



Gambar 1.4 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



Gambar 1.5 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

Name	Diamonds	Score	Time
stima	♥♥	0	43s
stima2	♥	0	43s
stima1	♥♥♥♥	0	44s
stima3	♥	0	44s

Gambar 1.6 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.

Untuk mengetahui *flow* dari game ini, berikut ini adalah cara kerja permainan Diamonds.

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 2

LANDASAN TEORI

2.1 Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara per langkah (*step by step*). Algoritma ini merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Persoalan optimasi (*optimization problems*) adalah persoalan mencari solusi optimal. Terdapat dua macam persoalan optimasi, yaitu maksimasi (*maximization*) dan minimasi (*minimization*).

Pada setiap langkah penyelesaiannya, algoritma ini menggunakan prinsip “*take what you can get now!*”, yaitu mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi kedepan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

2.1.1 Elemen Algoritma *Greedy*

Elemen-Elemen yang terdapat pada Algoritma *Greedy*:

1. Himpunan Kandidat (C)

Elemen ini berisi kandidat yang akan dipilih pada setiap langkah.

2. Himpunan Solusi (S)

Elemen ini berisi kandidat yang sudah dipilih

3. Fungsi Solusi

Elemen ini berupa fungsi yang akan menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi atau tidak.

4. Fungsi Seleksi (*Selection Function*)

Elemen ini berupa fungsi yang akan memilih kandidat berdasarkan strategi *greedy* tertentu (strategi *greedy* ini bersifat heuristik).

5. Fungsi Kelayakan (*feasible*)

Elemen ini berupa fungsi yang akan memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)

6. Fungsi Objektif

Elemen ini akan memaksimumkan dan memminimumkan.

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa Algoritma *greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S

dioptimisasi oleh fungsi objektif. Apabila dalam skema *pseudocode*, skema umum dari algoritma *greedy* dapat dilihat pada gambar 2.1.1 berikut.

```

function greedy( $C : \text{himpunan\_kandidat}$ )  $\rightarrow$   $\text{himpunan\_solusi}$ 
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
 $x : \text{kandidat}$ 
 $S : \text{himpunan\_solusi}$ 

Algoritma:
 $S \leftarrow \{\}$  { inisialisasi  $S$  dengan kosong }
while (not SOLUSI( $S$ )) and ( $C \neq \{\}$ ) do
     $x \leftarrow \text{SELEKSI}(C)$  { pilih sebuah kandidat dari  $C$ }
     $C \leftarrow C - \{x\}$  { buang  $x$  dari  $C$  karena sudah dipilih }
    if LAYAK( $S \cup \{x\}$ ) then {  $x$  memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
         $S \leftarrow S \cup \{x\}$  { masukkan  $x$  ke dalam himpunan solusi }
    endif
endwhile
{SOLUSI( $S$ ) or  $C = \{\}$ }

if SOLUSI( $S$ ) then { solusi sudah lengkap }
    return  $S$ 
else
    write('tidak ada solusi')
endif

```

Gambar 2.1 Skema Umum Algoritma Greedy

(Sumber: [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)-Bag1.pdf))

Pada akhir setiap iterasi, solusi yang terbentuk adalah optimum lokal dan pada akhir kalang while-do diperoleh optimum global (jika ada). Namun, optimum global belum tentu merupakan solusi optimum (terbaik) karena bisa jadi merupakan solusi sub-optimum atau pseudo-optimum.

Alasan hal tersebut bisa terjadi adalah:

1. Algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada (sebagaimana pada metode *exhaustive search*).
2. Terdapat beberapa fungsi seleksi yang berbeda sehingga kita harus memilih fungsi yang tepat jika kita tidak ingin algoritma menghasilkan solusi optimal.

Jadi, algoritma *greedy* tidak selalu berhasil memberikan solusi yang optimal, melainkan suboptimal pada sebagian persoalan.

Namun, jika solusi terbaik mutlak tidak selalu diperlukan, maka algoritma *greedy* dapat digunakan untuk menghasilkan solusi hampiran (*approximation*) daripada menggunakan algoritma yang kebutuhan waktunya eksponensial ataupun faktorial untuk menghasilkan solusi yang eksak.

Namun, bila algoritma *greedy* dapat menghasilkan solusi optimal, keoptimalannya itu harus dapat dibuktikan secara sistematis yang dapat menjadi tantangan tersendiri, sehingga dapat disimpulkan

bahwa mencari ketidakoptimalan algoritma *greedy* jauh lebih mudah dengan menunjukkan *counterexample* (contoh kasus yang menunjukkan solusi yang diperoleh tidak optimal).

2.1.2 Contoh-Contoh Persoalan yang Diselesaikan dengan Algoritma Greedy

1. Persoalan penukaran uang (*coin exchange problem*)
2. Persoalan memilih aktivitas (*activity selection problem*)
3. Minimisasi waktu di dalam sistem
4. Persoalan knapsack (*knapsack problem*)
5. Penjadwalan job dengan tenggat waktu (*job scheduling with deadlines*)
6. Pohon merentang minimum (*minimum spanning tree*)
7. Lintasan terpendek (*shortest path*)
8. Kode Huffman (*Huffman code*)
9. Pecahan Mesir (*Egyptian fraction*)

2.2 Cara Kerja Program Secara Umum

Permainan *Diamonds* ini terdiri dari dua komponen utama yaitu *game engine* dan *bot starter pack*. *Game engine* berisi kode *backend* permainan, yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *frontend* dan program bot. Selain itu, *game engine* juga berisi kode *frontend* yang berinteraksi langsung dengan pemain, menangani tampilan grafis, antarmuka pengguna, dan elemen-elemen visual lainnya yang memungkinkan pemain berinteraksi dengan permainan. Sementara itu, *bot starter pack* secara umum berisi program untuk memanggil API yang tersedia pada *backend*. Pada bagian ini juga terdapat program bot *logic* yang akan diimplementasikan dengan algoritma *greedy*. *Bot starter pack* memuat sekumpulan instruksi yang diperlukan untuk memulai pengembangan atau penggunaan bot dalam permainan *Diamonds*.

Permainan ini merupakan permainan berbasis *web* dengan *frontend* menggunakan *localhost* melalui <http://localhost:8082/> sehingga setiap aksi yang dilakukan – mulai dari mendaftarkan bot hingga menjalankan aksi bot – akan memerlukan HTTP *request* terhadap API *endpoint* tertentu yang disediakan oleh *backend*. Pada saat *game engine* dinyalakan, maka urutan *request* yang terjadi dari awal permulaan permainan adalah sebagai berikut.

1. Pengecekan Registrasi Bot

Program pertama kali akan melakukan pengecekan apakah bot sudah terdaftar atau belum dengan mengirimkan permintaan POST ke endpoint /api/bots/recover. Permintaan ini berisi email dan password bot. Jika bot sudah terdaftar, backend akan memberikan respons dengan kode 200 dan menyertakan id bot dalam body respons. Jika belum terdaftar, respons akan berupa kode 404.

2. Registrasi Bot Baru

Jika bot belum terdaftar, program bot akan mengirimkan permintaan POST ke endpoint `/api/bots` dengan body berisi email, nama, password, dan tim. Jika registrasi berhasil, backend akan memberikan respons dengan kode 200 dan menyertakan id bot dalam body respons.

3. Bergabung ke *Board*

Setelah mendapatkan id bot, program bot dapat bergabung ke *board* dengan mengirimkan permintaan POST ke endpoint `/api/bots/{id}/join`. Body permintaan berisi id papan yang diinginkan (`preferredBoardId`). Jika bot berhasil bergabung, backend akan merespons dengan kode 200 dan menyertakan informasi dari board dalam body respons.

4. Pergerakan Bot

Program bot secara berkala akan menghitung langkah selanjutnya berdasarkan kondisi papan yang diketahui dan mengirimkan permintaan POST ke endpoint `/api/bots/{id}/move`. Body permintaan berisi arah pergerakan (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Jika berhasil, backend akan memberikan respons dengan kode 200 dan menyertakan kondisi terbaru dari papan dalam *body respons*. Instruksi ini terus diulang hingga waktu bot habis, dan jika waktu habis, bot otomatis dikeluarkan dari *board*.

5. Update Kondisi Board pada Frontend

Tampilan *board* pada *frontend* selalu menunjukkan kondisi ter-update sesuai dengan kondisi terkini. Dalam hal ini, program *frontend* secara periodik akan mengirimkan permintaan GET ke endpoint `/api/boards/{id}` untuk mendapatkan kondisi terbaru dari *board*.

Untuk memulai permainan *Diamonds* dapat dilakukan dengan mengikuti alur cara menjalankan permainan *Diamonds* sebagai berikut.

1. Menjalankan *Game Engine*

a. *Requirement* yang harus di-*install*

- Node.js (<https://nodejs.org/en>)
- Docker desktop (<https://www.docker.com/products/docker-desktop/>)
- Yarn

```
npm install --global yarn
```

b. Instalasi dan konfigurasi awal

- 1) Download source code (.zip) pada [release game engine](#)
- 2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal
- 3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-game-engine-1.1.0
```

- 4) Install dependencies menggunakan Yarn

```
yarn
```

- 5) Setup default environment variable dengan menjalankan script berikut

Untuk Windows

```
./scripts/copy-env.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/copy-env.sh  
./scripts/copy-env.sh
```

- 6) Setup local database (buka aplikasi docker desktop terlebih dahulu, lalu jalankan command berikut di terminal)

```
docker compose up -d database
```

Lalu jalankan script berikut. Untuk Windows

```
./scripts/setup-db-prisma.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/setup-db-prisma.sh  
./scripts/setup-db-prisma.sh
```

c. *Build*

```
npm run build
```

d. *Run*

```
npm run start
```

Setelah berhasil, kunjungi *frontend* melalui <http://localhost:8082/>.

2. Menjalankan *Bot*

a. *Requirement* yang harus di-install

- Python (<https://www.python.org/downloads/>)

b. Instalasi dan konfigurasi awal

- 1) Download source code (.zip) pada [release bot starter pack](#)

- 2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal

- 3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-bot-starter-pack-1.0.1
```

- 4) Install dependencies menggunakan pip

```
pip install -r requirements.txt
```

c. *Run*

Untuk menjalankan satu bot (pada contoh ini, kita menjalankan satu bot dengan logic yang terdapat pada file game/logic/random.py)

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

Untuk menjalankan beberapa bot sekaligus (pada contoh ini, kita menjalankan 4 bot dengan logic yang sama, yaitu game/logic/random.py)

- Untuk windows

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

- Untuk Linux / (possibly) macOS

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

BAB 3

APLIKASI STRATEGI GREEDY

3.1 Mapping Diamonds secara Umum

Sebelum menguraikan berbagai alternatif solusi yang ada, kami mengelompokan persoalan *Diamonds* tersebut ke dalam elemen-elemen algoritma secara umum sebagai berikut.

Tabel 3.1.1: Mapping Persoalan Diamond secara Umum

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan beberapa aksi yang dilakukan pemain (akan dijelaskan lebih lanjut pada sub persoalan).
Himpunan Solusi	Aksi pemain yang sesuai dengan keadaan saat itu.
Fungsi Solusi	Memeriksa apakah aksi terpilih termasuk aksi yang valid.
Fungsi Seleksi	Memilih aksi yang paling menguntungkan, yaitu aksi yang dapat memperoleh banyak diamond dan menghindari konflik dengan lawan (akan dijelaskan lebih lanjut pada sub persoalan).
Fungsi Kelayakan	Memeriksa apakah aksi layak dilakukan berdasarkan fungsi kelayakan (akan dijelaskan lebih lanjut pada sub persoalan).
Fungsi Objektif	Mengumpulkan <i>diamond</i> sebanyak-banyaknya dan menghindari konflik dengan <i>bot</i> lain.

3.2 Alternatif utama Algoritma *Greedy*

3.2.1 Algoritma *greedy* berdasarkan jarak *diamond* terdekat

Algoritma ini berfokus pada perpindahan terdekat dari *bot* ke *diamond* tanpa memperdulikan warna serta bobot *diamond*. Strategi ini tergolong sederhana dan cukup naif karena hanya mengimplementasikan ambil yang terdekat terlebih dahulu untuk saat ini, menjadikannya solusi optimum lokal, sebelum akhirnya melakukan hal serupa untuk langkah berikutnya.

1. Pemetaan Elemen *Greedy*

Tabel 3.2.1: Pemetaan Elemen Greedy untuk Solusi Jarak Diamond Terdekat

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan <i>diamond</i> yang terdapat pada <i>board</i> , meliputi <i>diamond</i> merah dengan bobot 2 poin dan <i>diamond</i> biru dengan bobot 1 poin.
Himpunan Solusi	Himpunan <i>diamond</i> yang dipilih.
Fungsi Solusi	Menjumlahkan poin berdasarkan <i>diamond</i> yang didapatkan.
Fungsi Seleksi	Memilih <i>diamond</i> dengan jarak terdekat dari bot.
Fungsi Kelayakan	Memeriksa apakah jumlah <i>diamond</i> yang diperoleh tidak melebihi kapasitas default <i>inventory</i> dan dijamin balik ke base dalam setiap iterasi pencarian yang dilakukan.
Fungsi Objektif	Memaksimumkan jumlah poin <i>diamond</i> yang diperoleh.

2. Analisis Efisiensi Solusi

Pada alternatif ini, akan dilakukan penghitungan jarak untuk setiap *diamond* dalam *board*.

Kemudian, *bot* akan bergerak menuju *diamond* dengan jarak terdekat. Iterasi *array* dilakukan pada *board* sebanyak satu kali sehingga kompleksitas algoritma solusi ini adalah O(n).

3. Analisis Efektivitas Solusi

Strategi ini menjadi efektif apabila menemui kondisi berikut :

- Terdapat portal yang membantu bot untuk menuju *diamond* dan/atau kembali ke *base*.
- Bot muncul pada daerah dengan banyaknya diamond per area yang tinggi. Hal ini dapat membuat bot dengan cepat mengumpulkan *diamond* dan kembali ke base lebih cepat.

Strategi ini menjadi tidak efektif apabila menemui kondisi berikut :

- Terdapat *diamond* merah yang jaraknya memang sedikit lebih jauh dibandingkan *diamond* biru, tetapi jika diambil dapat lebih efektif.
- Portal menghalangi langkah *bot* sehingga harus melakukan langkah tambahan lebih banyak.
- Bot *spawn* pada daerah dengan jumlah *diamond* yang sedikit, sehingga *bot* menghabiskan waktu untuk menuju *diamond* dan kembali menyetor *diamond* ke *base*.

3.2.2 Algoritma *greedy* berdasarkan bobot *diamond* terbesar

Algoritma ini berfokus pada bobot dari *diamond* yang dapat diperoleh jika diambil tanpa memperdulikan jarak *diamond* dari posisi bot saat ini.

1. Pemetaan Elemen *Greedy*

Tabel 3.2.2: Pemetaan Elemen Greedy untuk Solusi Bobot Diamond Terbesar

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan <i>diamond</i> yang terdapat pada <i>board</i> , meliputi <i>diamond</i> merah dengan bobot 2 poin dan <i>diamond</i> biru dengan bobot 1 poin.
Himpunan Solusi	Himpunan <i>diamond</i> yang dipilih.
Fungsi Solusi	Menjumlahkan poin berdasarkan <i>diamond</i> yang didapatkan.
Fungsi Seleksi	Memilih <i>diamond</i> dengan memprioritaskan bobot maksimum tanpa mempertimbangkan jarak.
Fungsi Kelayakan	Memeriksa apakah jumlah <i>diamond</i> yang diperoleh tidak melebihi kapasitas <i>default inventory</i> dan dijamin balik ke <i>base</i> dalam setiap iterasi pencarian yang dilakukan.
Fungsi Objektif	Memaksimumkan jumlah poin <i>diamond</i> yang diperoleh.

2. Analisis Efisiensi Solusi

Pada alternatif ini, akan dicari jarak dari tiap *diamond*. Namun *bot* akan memprioritaskan untuk pergi ke *diamond* merah. Kompleksitas algoritma untuk solusi ini adalah O(n) karena terdapat iterasi *array* sebanyak satu kali.

3. Analisis Efektivitas Solusi

Strategi ini dapat menjadi efektif apabila menemui kondisi berikut :

- Posisi *spawn bot* memiliki konsentrasi *diamond* merah yang tinggi.
- Jarak bot ke *diamond* merah relatif pendek-pendek.
- Terdapat portal yang membantu bot untuk menuju *diamond* dan/atau kembali ke *base*.

Strategi ini dapat menjadi tidak efektif apabila menemui kondisi berikut :

- *Diamond* merah yang tersebar cukup sedikit sehingga poin yang dapat diperoleh pun menjadi lebih rendah.
- Posisi *Diamond* merah yang cukup jauh dari posisi *bot* saat ini sehingga membuang-buang waktu untuk mengambil *Diamond* merah meskipun sebetulnya terdapat kumpulan *diamond* biru yang jaraknya lebih dekat.

3.2.3 Algoritma *greedy* berdasarkan bobot per *move* terbesar

Algoritma ini berfokus pada perpindahan terdekat dari bot ke *diamond* dengan mempertimbangkan bobot *diamond*.

1. Pemetaan Elemen *Greedy*

Tabel 3.2.3: Pemetaan Elemen Greedy untuk Solusi Bobot per move Terbesar

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan <i>diamond</i> yang terdapat pada <i>board</i> , meliputi <i>diamond</i> merah dengan bobot 2 poin dan <i>diamond</i> biru dengan bobot 1 poin.
Himpunan Solusi	Himpunan <i>diamond</i> yang dipilih.
Fungsi Solusi	Menjumlahkan poin berdasarkan <i>diamond</i> yang didapatkan.
Fungsi Seleksi	Memilih <i>diamond</i> dengan bobot per <i>move</i> terbesar.
Fungsi Kelayakan	Memeriksa apakah jumlah <i>diamond</i> yang diperoleh tidak melebihi kapasitas <i>default</i>

	<i>inventory</i> dan dijamin balik ke base dalam setiap iterasi pencarian yang dilakukan.
Fungsi Objektif	Memaksimumkan jumlah poin <i>diamond</i> yang diperoleh.

2. Analisis Efisiensi Solusi

Pada alternatif ini, akan dicari jarak dari tiap *diamond*. Kali ini *bot* akan memprioritaskan untuk pergi ke *diamond* dengan poin per *move* yang paling tinggi sehingga *bot* tidak selalu menuju *diamond* merah walaupun terdapat *diamond* merah. Kompleksitas solusi ini juga O(n) karena hanya diperlukan satu kali iterasi pada array *diamond* yang ada pada *board*.

3. Analisis Efektivitas Solusi

Strategi ini efektif apabila menemui kondisi berikut :

- Jarak *base* kita dengan *base* musuh tidak berdekatan sehingga meminimalisir kemungkinan untuk saling bertabrakan.
- Daerah di sekitar posisi *spawn* memiliki konsentrasi *diamond* yang tinggi sehingga *bot* dapat mengambil *diamond* sesegera mungkin dan kembali ke *base* dengan cepat.
- Terdapat portal yang membantu bot untuk menuju *diamond* dan/atau kembali ke *base*.
- Terdapat bot lawan yang mengejar atau terletak satu blok darinya. Hal ini dapat memicu *trigger* pada bot untuk bertabrakan dengan bot lawan.

Namun, strategi ini tidak efektif apabila menemui kondisi berikut :

- Bot tidak sempat kembali setelah mengumpulkan *diamond*.
- Bot menuju ke daerah yang konsentrasi *diamondnya* rendah.

3.2.4 Algoritma *greedy* berdasarkan jarak *diamond* dengan *base*

Algoritma ini berfokus pada perpindahan terdekat dari *base* ke *diamond* tanpa memperdulikan warna *diamond* dan jarak bot dengan *diamond*.

1. Pemetaan Elemen *Greedy*

Tabel 3.2.4: Pemetaan Elemen Greedy untuk Solusi Jarak Diamond dengan Base

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan <i>diamond</i> yang terdapat pada <i>board</i> , meliputi <i>diamond</i> merah dengan bobot 2 poin dan <i>diamond</i> biru dengan bobot 1 poin.

Himpunan Solusi	Himpunan diamond yang dipilih.
Fungsi Solusi	Menjumlahkan poin berdasarkan diamond yang didapatkan.
Fungsi Seleksi	Memilih diamond dengan jarak terdekat dari base.
Fungsi Kelayakan	Memeriksa apakah jumlah diamond yang diperoleh tidak melebihi kapasitas default inventory dan dijamin balik ke base dalam setiap iterasi pencarian yang dilakukan.
Fungsi Objektif	Memaksimumkan jumlah poin diamond yang diperoleh.

2. Analisis Efisiensi Solusi

Bot akan menghitung jarak semua diamond ke base. Bot kemudian memilih untuk pergi ke diamond yang terletak dekat dengan base. Kompleksitasnya juga $O(n)$ karena hanya diperlukan iterasi satu kali pada array.

3. Analisis Efektivitas Solusi

Strategi ini menjadi efektif apabila memenuhi kondisi berikut

- Jarak base kita dengan base musuh tidak berdekatan sehingga meminimalisir kemungkinan untuk saling bertabrakan.
- Daerah di sekitar posisi *spawn* memiliki konsentrasi diamond yang tinggi sehingga bot dapat mengambil *diamond* sesegera mungkin dan kembali ke base dengan cepat.
- Terdapat portal yang membantu bot untuk menuju *diamond* dan/atau kembali ke *base*.

Strategi ini menjadi tidak efektif apabila memenuhi kondisi berikut

- Diamond tersebar di base secara merata sehingga bot harus bolak-balik untuk mengumpulkan diamond yang paling dekat base.
- Jarak antara *diamond* satu ke *diamond* lainnya yang besar meskipun berada di sekitar base sehingga membuat bot melakukan pergerakan yang besar.

3.2.5 Algoritma greedy berdasarkan konsentrasi diamond dalam suatu daerah

Algoritma ini berfokus pada suatu daerah pada board dengan konsentrasi diamond paling banyak.

1. Pemetaan Elemen Greedy

Tabel 3.2.5: Pemetaan Elemen Greedy untuk Konsentrasi Diamond dalam suatu Daerah

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan diamond yang terdapat pada board, meliputi diamond merah dengan bobot 2 poin dan diamond biru dengan bobot 1 poin.
Himpunan Solusi	Himpunan diamond yang dipilih.
Fungsi Solusi	Menjumlahkan poin berdasarkan diamond yang didapatkan.
Fungsi Seleksi	Memilih diamond dengan konsentrasi terbesar.
Fungsi Kelayakan	Memeriksa apakah jumlah diamond yang diperoleh tidak melebihi kapasitas default inventory dan dijamin balik ke base dalam setiap iterasi pencarian yang dilakukan.
Fungsi Objektif	Memaksimumkan jumlah poin diamond yang diperoleh.

2. Analisis Efisiensi Solusi

Pada solusi ini, program akan membagi map menjadi 4 partisi daerah. Bot akan menghitung konsentrasi diamond pada tiap daerah dengan cara menghitung jumlah diamond pada daerah itu dibagi dengan jaraknya dengan bot. Kompleksitasnya juga $O(n)$.

3. Analisis Efektivitas Solusi

Strategi ini efektif jika menemui kondisi berikut :

- Bot *spawn* pada daerah dengan konsentrasi diamond tinggi sehingga dapat mengumpulkan diamond kembali ke base dengan cepat.
- Terdapat portal yang membantu bot untuk menuju daerah dengan konsentrasi tinggi dan/atau pulang ke *base*.

Strategi ini tidak efektif jika menemui kondisi berikut :

- Bot lawan terdapat pada daerah dengan konsentrasi *diamond* tinggi, sehingga bot kalah cepat untuk mengambil *diamond*.
- Bot memilih daerah dengan konsentrasi diamond tinggi, tetapi jaraknya sangat jauh dari *base* sehingga keseluruhan pengumpulan dan penyetoran *diamond* memakan waktu yang lama.
- Bot mengambil *diamond* dari suatu partisi daerah mengakibatkan konsentrasi daerah itu turun. Hasilnya, bot menuju daerah lain yang mungkin lebih jauh.

3.2.6 Algoritma greedy berdasarkan inventory lawan

Algoritma ini berfokus untuk men-tackle lawan yang memiliki inventory yang paling banyak.

1. Pemetaan Elemen Greedy

Tabel 3.2.6: Pemetaan Elemen Greedy untuk Solusi Mencari Lawan dengan diamond di Inventory Paling Besar

Nama Elemen	Definisi Elemen
Himpunan kandidat	Himpunan lawan yang ikut bermain pada permainan.
Himpunan Solusi	Himpunan diamond yang direbut dari pihak lawan
Fungsi Solusi	Menjumlahkan poin berdasarkan diamond yang didapatkan.
Fungsi Seleksi	Memilih lawan dengan perolehan diamond pada inventory terbesar.
Fungsi Kelayakan	Memeriksa apakah jumlah diamond yang diperoleh tidak melebihi kapasitas default inventory dan dijamin balik ke base dalam setiap iterasi pencarian yang dilakukan.
Fungsi Objektif	Memaksimumkan jumlah poin diamond yang diperoleh.

2. Analisis Efisiensi Solusi

Pada solusi ini, bot akan mencari jumlah inventory dari semua lawan. Bot kemudian memilih untuk menunggu pada base lawan yang mempunyai banyak diamond pada *inventory*-nya. Kompleksitas pada solusi ini adalah O(n).

3. Analisis Efektivitas Solusi

Strategi ini efektif apabila memenuhi kondisi berikut

- Bot musuh dekat dengan kita sehingga dapat di-*tackle*
- Bot men-*tackle* ketika dekat dengan base sehingga bisa menyetor diamond dengan cepat

Strategi ini tidak efektif apabila memenuhi kondisi berikut

- Efek *boomerang*. Hal ini terjadi ketika kita ter-*tackle* (baik disengaja atau tidak) oleh musuh, yang menyebabkan bot menyimpang dari tujuan awal, yaitu menunggu musuh di base mereka.
- Bot musuh tidak sempat kembali ke basenya karena di-*tackle* oleh musuh lain. Hal ini menyebabkan bot justru tidak mempunyai poin.

3.3 Strategi Greedy yang diimplementasikan

Berdasarkan alternatif-alternatif algoritma greedy yang telah dijabarkan pada bagian sebelumnya, diputuskan bahwa strategi yang akan digunakan adalah strategi greedy dengan bobot per move terbesar. Alasan penggunaan strategi ini adalah membuat bot lebih konsisten dalam membuat keputusan per move, seperti berfokus pada mengambil sebanyak-banyaknya diamond yang ada. Dengan kata lain, rumus untuk menentukan densitas pada solusi ini adalah :

$$\text{Densitas} = \text{bobot} / \text{jarak}$$

Jarak pada rumus ini merupakan jumlah move yang diperlukan untuk mencapai objek yang dimasud. Bobot untuk diamond biru adalah 1 dan untuk diamond merah adalah 2. Bot akan kembali ke base ketika inventorynya sudah penuh dan tidak akan mengambil diamond merah jika inventorynya sudah terisi 4 diamond.

Strategi tersebut dapat dioptimalkan dengan menambahkan beberapa strategi pendukung, yaitu:

- Mempertimbangkan penggunaan teleport sebagai salah satu langkah dalam melakukan optimasi pencarian *diamond*. Pada setiap diamond akan dicari alternatif total *move* yang diperlukan dari posisi bot jika melewati teleporter. Dalam hal ini perhitungan total *move* dilakukan sedemikian menjadi:

$$\text{total move} = d1 + d2, \text{ dengan}$$

d1: jarak bot ke teleporter terdekat

d2: jarak teleporter hasil teleportasi ke *diamond* yang dituju

- Memperhitungkan bobot *red button*, yaitu memberikan bobot *red button* dengan konstanta $4/9$. Konstanta ini diperoleh dari eksperimen perbandingan antara total *move* bot ke *red button* dengan total *move* bot ke *diamond*. Dalam hal ini, artinya jika total *move* dari bot ke *red button* adalah 4, sementara total *move* dari bot ke *diamond* adalah 9 atau lebih, maka bot akan menyentuh *red button*. Hal ini membuat papan permainan melakukan *re-generate* kembali posisi *diamond* yang ada sehingga diharapkan akan memberikan posisi *diamond* yang lebih menguntungkan dibanding kondisi sebelumnya.
- Untuk menghindari lawan, bot berjalan menggunakan pola *zig-zag*. Pola ini dilakukan setelah selisih antara sumbu x dan sumbu y dari posisi bot menuju *goal position* dalam keadaan simetri (persegi).
- Pertimbangan terhadap sisa waktu sebelum permainan berakhir. Bot akan kembali ke *base* jika waktu *bot* untuk kembali ke *base* tepat dengan waktu tersisa permainan sebelum berakhir. Dengan demikian, *diamond* yang ada pada *inventory* di akhir waktu sempat untuk diubah menjadi poin. Pada saat kembali menuju *base*, jika bot masih memiliki ruang untuk *diamond* pada *inventory*, maka bot akan berjalan menuju *base* sembari mencari *diamond*. Pencarian ini dilakukan tanpa menambah total *move* yang diperlukan untuk kembali ke *base* sehingga menghindari keterlambatan untuk sampai ke *base* dan menyelamatkan koleksi *diamond* di akhir sisa waktu.
- Bot akan kembali ke *base* jika memang jalur yang dilalui melalui *base* tidak akan mengurangi total *move* untuk ke objek yang ingin dituju. Dengan demikian, bot akan lebih aman dari *tackle* bot lain dan *inventory*nya kosong lagi sehingga bisa mengambil lebih banyak *diamond*.
- Ketika *inventory*nya sudah 4, jarak yang dipakai untuk mencari densitas adalah jarak *bot* ke *diamond* ditambah jarak *diamond* ke *base*. Hal ini membuat bot memiliki total *move* yang optimal untuk kembali ke *base* ketika *inventory* telah penuh, menghindari kemungkinan membawa banyak *diamond* berkeliling pada *board* dan mendapat *tackle* dari bot lain.
- Bot tidak akan mengambil *diamond* dengan point 2 ketika total *diamond* di *inventory* adalah 4, menghindari *error* terhadap pengambilan *red diamond* yang gagal dan menyebabkan bot dalam kondisi *invalid move* diam di tempat (0, 0).
- Menghindari teleporters jika menimbulkan kerugian. Ketika langkah berikutnya dari bot adalah menuju kotak yang berisi teleporter, sementara teleporter membuat bot berpindah menjauhi *goal position* yang telah ditentukan sebelumnya, maka bot akan menghindar dari kotak yang berisi teleporter. Hal ini akan menghindarkan bot dari kerugian total *move* yang semakin besar.

Alasan alternatif algoritma *greedy* berdasarkan jarak terdekat ke *diamond* tidak dipilih karena terlalu sederhana dalam pendekatan pengumpulan *diamond*. Kriteria ini tidak mempertimbangkan faktor-faktor tambahan yang dapat mempengaruhi strategi yang lebih optimal dalam mengambil

diamond. Selanjutnya, alasan alternatif *greedy* berdasarkan bobot *diamond* terbesar juga tidak dipilih karena tidak sejalan dengan tujuan utama, yaitu memaksimumkan perolehan *diamond* secara keseluruhan. Meskipun mungkin menarik untuk mengejar *diamond* dengan bobot terbesar, namun dapat terjadi kemungkinan bahwa mengumpulkan *diamond* dengan bobot yang lebih rendah tetapi dalam jumlah besar dapat lebih menguntungkan secara keseluruhan.

Pendekatan berdasarkan konsentrasi terbesar juga dihindari karena kurang konsisten dalam mendapatkan *diamond*. Situasi di mana area dengan konsentrasi tinggi terlalu jauh untuk dijangkau atau sudah diambil oleh lawan menambah keraguan terhadap pendekatan ini. Hal yang sama berlaku untuk pendekatan berdasarkan *base* terdekat, yang memiliki masalah serupa dengan ketidakpastian ketersediaan *diamond* di sekitar *base* lawan.

Alasan alternatif *greedy* berdasarkan *base* terdekat tidak dianggap sebagai pilihan yang optimal karena memiliki masalah yang serupa dengan pendekatan berdasarkan konsentrasi terbesar. Dalam kedua kasus, terdapat ketidakpastian dalam ketersediaan *diamond* di sekitar *base* lawan, serta kemungkinan bahwa area dengan *base* terdekat atau konsentrasi terbesar sudah diambil oleh lawan.

Selain itu, alasan alternatif *greedy* berdasarkan *inventory* lawan juga tidak dipilih karena memiliki risiko yang cukup tinggi. Dengan mempertimbangkan *inventory* lawan, terdapat kemungkinan besar untuk ter-*tackle* oleh lawan, baik secara sengaja maupun tidak. Oleh karena itu, strategi ini dianggap tidak layak karena berpotensi mengakibatkan kerugian besar bagi pemain.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi dengan pseudocode

Berikut implementasi yang mendasari pergerakan dari bot.

```
Function direction (current_x: integer, current_y: integer,
                    dest_x: integer, dest_y: integer) → (integer, integer)
{ Function direction akan mengembalikan tuple (integer, integer),
  yaitu posisi yang mungkin dilakukan oleh bot untuk melakukan move
  (NORTH, SOUTH, EAST, atau WEST) menuju ke goal_position }
```

Kamus

delta_x, delta_y: integer

Algoritma

```
delta_x ← current_x - dest_x
delta_y ← current_y - dest_y
if abs(delta_x) > abs(delta_y) then
    if delta_x < 0 then
        → (1, 0)
    else
        → (-1, 0)
else
    if delta_y < 0 then
        → (0, 1)
    else
        → (0, -1)
```

```
Function calculate_move(current: Position, other: Position) → integer
{ Function calculate_move akan mengembalikan integer jumlah move yang
  diperlukan dari posisi current ke posisi other. Setiap berpindah dari satu
  sel ke sel lain terhitung satu satuan }
```

Algoritma

→ abs(current.x - other.x) + abs(current.y - other.y)

```
Function sorted_portal(current: Position,
                        board: Board) → (Position, Position)
{ Function sorted_portal akan mengembalikan tuple (Position, Position),
yaitu posisi portal yang telah terurut total movenya dari yang paling dekat
ke posisi bot, pengurutan dilakukan ascending }
```

Kamus

portal_1: Position
 portal_2: Position

```
portal_1 ← calculate_move(current,
                           board.game_objects[0].position)
portal_2 ← calculate_move(current,
                           board.game_objects[1].position)
if (portal_1 < portal_2) then
    portal_1 ← board.game_objects[0].position
    portal_2 ← board.game_objects[1].position
else
    portal_1 ← board.game_objects[1].position
    portal_2 ← board.game_objects[0].position
→ (portal_1, portal_2)
```

```
Function avoid_portal(board: Board, x: integer, y: integer) → bool
{ Function avoid_portal akan mengembalikan boolean true jika x dan y
merupakan posisi dari salah satu portal pada board, dan akan mengembalikan
false jika sebaliknya }
```

Kamus

portal_1: Position
 portal_2: Position

Algoritma

```
{ koordinat portal 1 }
xPortal_1 ← board.game_objects[0].position.x
yPortal_1 ← board.game_objects[0].position.y
{ koordinat portal 2 }
xPortal_2 ← board.game_objects[1].position.x
yPortal_2 ← board.game_objects[1].position.y
→ (xPortal_1 = x and yPortal_1 = y)
or (xPortal_2 = x and yPortal_2 = y)
```

```
Function destination(denseDiamond: integer, temp: integer,
    densePortal: integer, diamond: Position, portal: Position) → integer
{ Function destination akan mengembalikan integer density, yaitu hasil dari
poin per move. Move dihitung sebagai jumlah move dari diamond ke current
position }
```

Algoritma

```
if (temp <= denseDiamond) and (temp >= densePortal) then
    → temp
else
    if (denseDiamond > temp) then
        goal_position ← diamond
        passPortal ← False
        temp ← denseDiamond
    if (densePortal > temp) then
        goal_position ← portal
        passPortal ← True
        temp ← densePortal
    → temp
```

```
Function next_move(board_bot: GameObject,
                    board: Board) → (integer, integer)
{ Function next_move akan mengembalikan tuple (integer, integer), yaitu
posisi move berikutnya yang mungkin dilakukan oleh bot (NORTH, SOUTH, EAST,
atau WEST) untuk menuju ke diamond, dengan mempertimbangkan beberapa hal
sesuai dengan algoritma greedy yang dipilih }
```

Algoritma

```
{ Inisialisasi }
props ← board_bot.properties
current ← board_bot.position

{ Cari teleport paling dekat }
portal_1, portal_2 ← sorted_portal(current, board)
passPortal ← False

timeLeft ← props.milliseconds_left
backBase ← calculate_move(current, props.base) *
```

```

board.minimum_delay_between_moves * 10
inventory ← props.diamonds

{ Kembali ke base jika diamond sudah 5 atau waktu hampir habis }
if inventory = 5 or (( backBase >= timeLeft - 3000
    and backBase < timeLeft) and not
        position_equals(current, props.base)) then
    goal_position ← None
    toBase ← calculate_move(current, props.base)

{ Kembali ke base sambil mencari diamond }
if inventory < 5 then
    diamond ← board.diamonds
    temp ← 0
    { Iterasi ke setiap diamond }
    { lalu cari diamond dengan poin/move paling besar }
    for d in diamond
        point ← d.properties.points
        takeDiamond ← calculate_move(current, d.position)
+
        calculate_move(d.position, props.base)
        distancePortal ← calculate_move(current, portal_1)
            + calculate_move(portal_2, d.position) +
            calculate_move(d.position, props.base)
    { Tidak bisa ambil red diamond }
    if inventory = 4 and point = 2 then
        continue
    else { Masih bisa ambil red diamond }
        denseDiamond ← point / takeDiamond
        densePortal ← point / distancePortal
        if toBase >= takeDiamond then
            temp ← destination(denseDiamond, temp, 0,
                d.position, portal_1)
        if toBase >= distancePortal then
            temp ← destination(0, temp, densePortal,
                d.position, portal_1)
    { Handle jika tidak ditemukan diamond, langsung pulang ke base }

```

```

{ Langsung kembali ke base }
if goal_position = None then
    portalToBase ← calculate_move(current, portal_1) +
        calculate_move(portal_2, props.base)
    if (portalToBase < toBase) then
        goal_position ← portal_1
        passPortal ← True
    else
        goal_position ← props.base

{ Mencari diamond (belum pulang ke base) }
else
    diamond ← board.diamonds
    distanceButton ← calculate_move(current,
        board.game_objects[2].position)
    temp ← 0
    { Inisialisasi red button sebagai goal_position }
    if distanceButton > 0 then
        temp ← (4 / 9) / distanceButton
        goal_position ← board.game_objects[2].position

    { Iterasi ke setiap diamond }
    { lalu cari diamond dengan poin/move paling besar }
    for d in diamond
        point ← d.properties.points
        distanceDiamond ← calculate_move(current, d.position)
        distancePortal ← (calculate_move(current, portal_1) +
            calculate_move(portal_2, d.position))
        if inventory <= 3 then { Masih bisa ambil red diamond }
            denseDiamond ← point / distanceDiamond
            densePortal ← point / distancePortal
            temp ← destination(denseDiamond, temp, densePortal,
                d.position, portal_1)
        { Tidak bisa ambil red diamond }
        elif (point != 2) then
            distanceBase ← calculate_move(d.position, props.base)
            denseDiamond ← point / (distanceDiamond + distanceBase)
            densePortal ← point / (distancePortal + distanceBase)
            temp ← destination(denseDiamond, temp, densePortal,

```

```

d.position, portal_1)

{ Jalan ke goal_position sambil men-store diamond ke base jika efektif }
toBase ← calculate_move(current, props.base) +
calculate_move(props.base, goal_position)
distanceDiamond ← calculate_move(current, goal_position)
if (toBased = distanceDiamond and not
position_equals(current, props.base)) then
goal_position ← props.base

{ cari direction dari destination }
delta_x, delta_y ← direction(
current.x, current.y,
goal_position.x, goal_position.y)

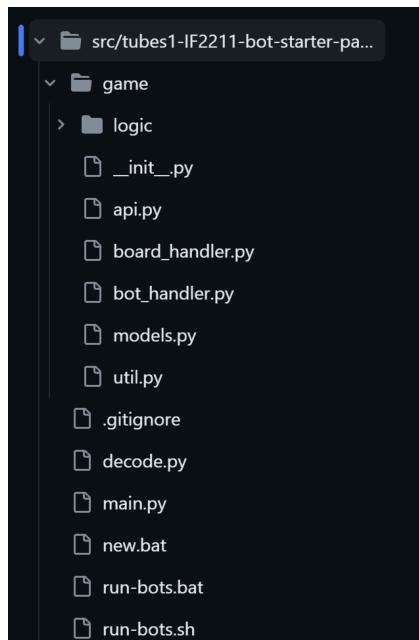
{ Hindari portal }
if avoid_portal(board, delta_x + current.x, delta_y
+ current.y) and not passPortal then
if delta_x != 0 then
if goal_position.y < current.y then
delta_x, delta_y ← (0, -1)
else
delta_x, delta_y ← (0, 1)
elif delta_y != 0 then
if goal_position.x < current.x then
delta_x, delta_y ← (-1, 0)
else
delta_x, delta_y ← (1, 0)

→ (delta_x, delta_y)

```

4.2 Struktur Data Program

Implementasi dilakukan dengan menggunakan bahasa pemrograman python. Folder src terdiri dari folder tubes1-IF2211-bot-starter-pack-1.0.1 yang berisi folder game, di dalamnya terdapat *folder logic, models* dari *game*, serta utilitas lainnya yang dipakai dalam implementasi *bot*. Struktur program lebih lanjut dapat dilihat pada gambar 4.2.1 berikut ini.



Gambar 4.2.1. Struktur Data Program

Struktur data pada program diimplementasikan dengan class. Class yang terdapat pada file *models.py* antara lain:

1. Class Bot

Class Bot ini digunakan untuk menginstansiasi objek *bot* dalam permainan Diamonds dan memiliki tiga atribut sebagai berikut.

- *name* : Nama objek *bot*.
- *email* : Alamat email objek *bot*.
- *id* : ID unik yang mengidentifikasi objek *bot*.

2. Class Position

Class Position ini menyimpan informasi mengenai koordinat x dan y. Kelas ini digunakan untuk merepresentasikan posisi objek.

3. Class Properties

Class Properties akan menyimpan informasi properti dari sebuah objek *bot*. *Properties* yang disimpan adalah atribut yang dimiliki oleh kelas ini, yaitu sebagai berikut.

- *points* : Jumlah poin yang diberikan oleh *diamond* (1 atau 2).

- *pair_id* : Informasi untuk melakukan *pairing* pada *Teleporter*.
- *diamonds* : Total *diamond* yang dipegang oleh *bot*.
- *score* : Poin total yang disimpan dalam *basenya* dan ditampilkan pada *scoreboard*.
- *name* : Nama objek *bot*.
- *inventory size* : Ukuran *inventory* yang dimiliki oleh objek (biasanya 5 untuk objek *bot*).
- *can tackle* : Informasi apakah objek bisa ditabrak atau tidak.
- *milliseconds_left* : Waktu tersisa untuk tetap bermain dalam papan permainan.
- *time_joined* : Waktu masuknya objek *bot* ke dalam permainan.
- *base* : Posisi letak *base* objek jika ada.

4. Class *GameObject*

Class GameObject menampung informasi seperti id, posisi, tipe, dan *properties* dari suatu objek. *Properties* pada atribut ini merupakan *extension class* dari '*Properties*' sebelumnya yang diinstansiasi menggunakan *blueprint* dari *class* ini.

5. Class *Board*

Class Board merupakan kelas yang mewakili papan permainan dalam *Diamonds*. Kelas ini menyimpan informasi atau pun *state* dari papan permainan Diamonds. Kelas ini memiliki atribut sebagai berikut.

- *id* : ID papan permainan.
- *lebar* : Lebar papan permainan.
- *tinggi* : Tinggi papan permainan.
- *features* : Fitur-fitur tambahan pada papan.
- *game_objects* : Berisi list *game object* yang ada pada papan
- *minimum_delay_between_moves* : Jeda minimum antar gerakan *bot*.

Pada file *util.py*, terdapat fungsi-fungsi pembantu yang dapat diimplementasikan untuk membantu program. Fungsi-fungsi itu adalah :

1. Fungsi *clamp*

Fungsi ini mengambil tiga parameter yaitu *n* (nilai yang akan di-*constraint*), *smallest* (batas nilai terkecil), dan *largest* (batas nilai terbesar). *Clamp* menggunakan fungsi *max* dan *min* untuk memastikan bahwa nilai *n* berada di antara batas nilai terkecil dan terbesar. Fungsi ini mengembalikan nilai *n* yang telah di-*constraint*.

2. Fungsi `get_direction`

Fungsi ini mengambil empat parameter yaitu `current_x` dan `current_y` (koordinat saat ini), serta `dest_x` dan `dest_y` (koordinat tujuan atau destinasi). Fungsi ini menghitung selisih antara koordinat tujuan dan saat ini untuk mendapatkan `delta_x` dan `delta_y`. Fungsi ini mengembalikan tuple (`delta_x, delta_y`) yang menunjukkan arah pergerakan bot, dengan menggunakan fungsi `clamp` untuk memastikan bahwa `delta_x` dan `delta_y` berada dalam rentang -1 hingga 1.

3. Fungsi `position_equals`:

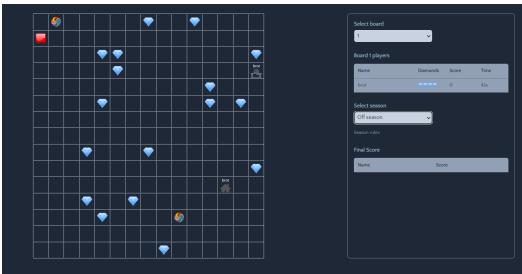
Fungsi ini memeriksa apakah koordinat x dan y dari kedua objek `position` sama atau tidak. Mengembalikan nilai `True` jika koordinatnya sama dan `False` jika sebaliknya.

Dalam folder `game/logic/`, terdapat file yang berisi logika implementasi alternatif algoritma *greedy*, yaitu pada file `botbang.py`. Logika ini berfungsi sebagai otak yang akan menentukan pergerakan permainan *bot*. Dalam mencari pergerakan *bot*, terdapat fungsi-fungsi bantuan yaitu `destination`, `avoid_portal`, `sorted_portal`, `calculate_move`, dan `direction`. Pergerakan *bot* merupakan tuple berisi delta x dan delta y yang di-return pada fungsi utama file ini yaitu `next_move()`. Selanjutnya file `main.py` pada folder `game` berisi algoritma utama yang menghubungkan semua komponen dalam `bot starter pack` ini. Logika yang telah diimplementasikan dapat diintegrasikan dengan menambahkannya ke dalam bagian `CONTROLLERS` yang terdapat dalam file "main.py". Dengan cara ini, logika tersebut menjadi bagian integral dari algoritma utama yang mengendalikan perilaku *bot* selama bermain dalam permainan Diamonds.

4.3 Analisis dan Pengujian

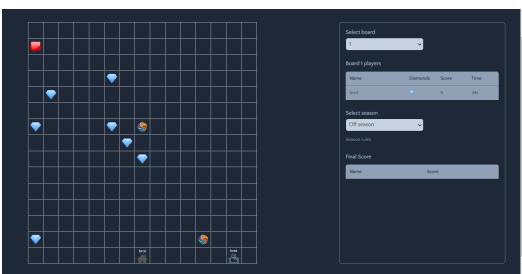
Tabel 4.3.1: Analisis dan Pengujian Program

No.	Screenshot	Penjelasan
1.	 Kondisi awal	Memperhitungkan red button : Pada awalnya, bot dalam keadaan mencari <i>diamond</i> dengan melakukan inisialisasi <i>red button</i> sebagai <i>goal position</i> saat ini dan nilai $(4/9) / (\text{total move}$ dari bot ke <i>red button</i>) sebagai <i>dense temporary</i> untuk dibandingkan pada tahap iterasi <i>diamond</i> . Hal tersebut dilakukan sebagai pertimbangan untuk bergerak menuju tombol <i>red button</i> jika total <i>move</i> terdekat dari bot ke <i>diamond</i> ≥ 9 , sementara total <i>move</i> terdekat dari bot ke <i>red button</i> ≤ 4 . Pada saat



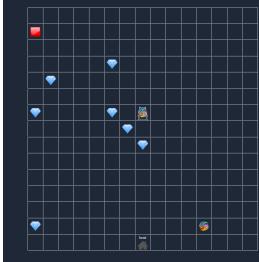
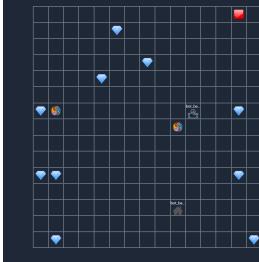
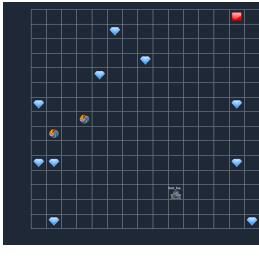
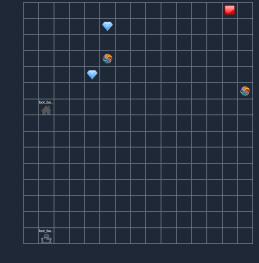
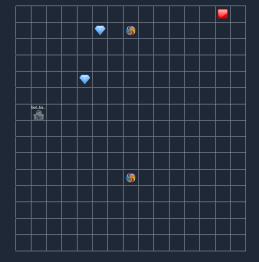
Kondisi akhir

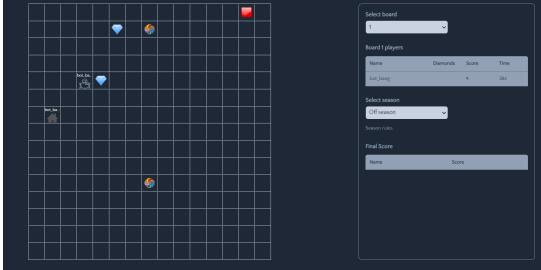
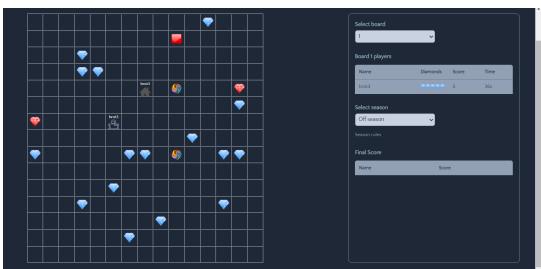
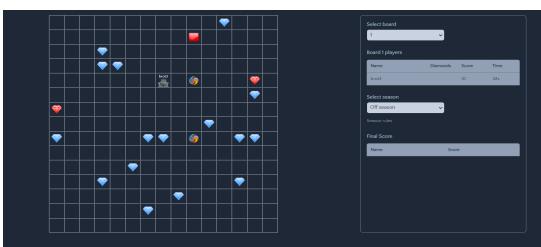
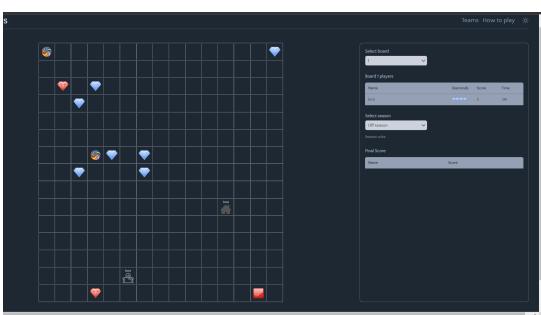
tahap iterasi setiap diamond, tidak ada kondisi di mana kombinasi nilai poin / (total move) yang memiliki nilai lebih besar dari pada *dense temporary*. Kondisi tersebut bertahan hingga iterasi diamond berakhir. Hal ini membuat *goal position* dari bot akan menuju ke *red button*. Papan permainan akan melakukan *re-generate* kembali posisi *diamond* yang ada. Kami memilih hal ini sebagai alternatif untuk mencari peruntungan karena jika bergerak menuju tempat *diamond* yang jauh terdapat kemungkinan akan kalah dengan bot yang lain. Selain itu, dengan melakukan *re-generate* akan membuat lawan yang sudah lebih dekat ke *diamond* yang sebelumnya menjadi harus mencari *direction* ke *diamond* yang lain. Namun, hal ini tidak efektif jika pada saat *re-generate diamond* menghasilkan *diamond* yang menguntungkan lawan dan merugikan bot sendiri. Implementasi yang diharapkan untuk persoalan ini akan memberikan posisi *diamond* yang setidaknya bisa lebih menguntungkan dibanding kondisi sebelumnya.

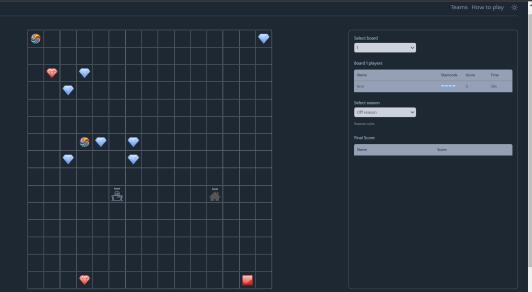
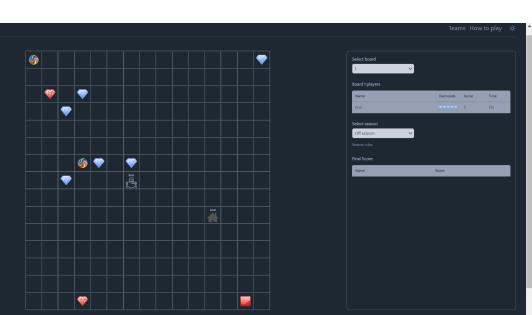


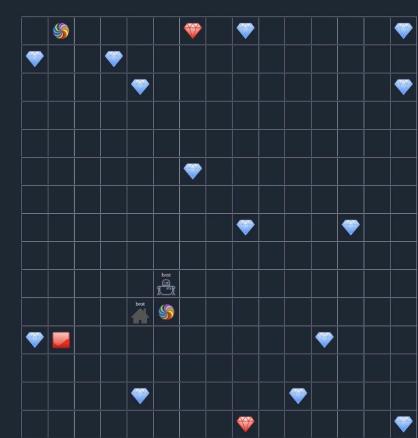
Kondisi awal

Manfaatkan portal : Pada awalnya, bot dalam keadaan mencari *diamond*. Pada saat tahap iterasi diamond, bot akan menghitung total move, baik secara langsung menuju *diamond* atau pun dengan penggunaan teleport sebagai salah satu langkah dalam melakukan optimasi pencarian *diamond*. Karena terdapat kondisi di mana pada saat melewati teleporter nilai poin / (total move) lebih besar dari pada *dense temporary*, maka

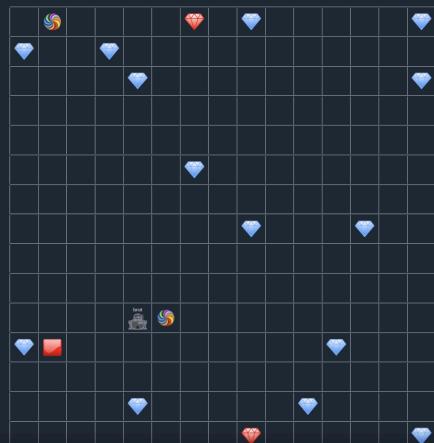
	 <p>Kondisi akhir</p>	<p>bot akan bergerak menuju ke teleporter terdekat untuk melakukan teleportasi. Selanjutnya, bot akan bergerak menuju <i>diamond</i>. Hal ini terbukti dapat memberikan total <i>move</i> yang optimal untuk bergerak ke <i>diamond</i>.</p>
3.	 <p>Kondisi awal</p>  <p>Kondisi akhir</p>	<p>Kondisi diamond di inventory ada 4 : Pada gambar di samping, meskipun <i>bot</i> masih memiliki 4 <i>diamond</i> di inventory dari total 5, <i>bot</i> secara otomatis kembali menuju ke <i>base</i> karena waktu permainan hanya menyisakan 10 detik. Hal ini menunjukkan bahwa salah satu strategi pendukung, yaitu pertimbangan terhadap sisa waktu sebelum permainan berakhir, telah diimplementasikan dengan baik.</p>
4.	 <p>Kondisi awal</p>  <p>Kondisi akhir</p>	<p>Menyetor diamond ke base selagi efektif : Jika terjadi kondisi di mana total <i>move</i> bot ke base ditambah dengan total <i>move</i> dari base ke <i>diamond</i> sama dengan total <i>move</i> bot ke <i>diamond</i> secara langsung, maka bot akan otomatis kembali ke base terlebih dahulu untuk mengamankan perolehan diamond sementara. Selanjutnya, bot akan bergerak kembali menuju <i>diamond</i> yang dituju. Hal ini merupakan implementasi dari strategi bahwa bot akan kembali ke <i>base</i> jika memang jalur yang dilalui melalui <i>base</i> tidak akan mengurangi total <i>move</i> untuk ke objek yang</p>

	<p>Kondisi tengah</p>  <p>Kondisi akhir</p>	<p>ingin dituju. Dengan demikian, bot akan lebih aman dari <i>tackle</i> bot lain dan <i>inventory</i>nya kosong lagi sehingga bisa mengambil lebih banyak <i>diamond</i>.</p>
5.	 <p>Kondisi awal</p>  <p>Kondisi akhir</p>	<p>Kondisi diamond sudah 5 : Pada gambar tersebut terlihat <i>inventory</i> pada bot sudah penuh (<i>diamond</i> = 5). Untuk mengamankan <i>diamond</i>, maka bot akan sesegera mungkin mencari jalan dengan total <i>move</i> paling sedikit menuju <i>base</i>. Implementasi ini adalah sebagai alternatif utama untuk kondisi bot melakukan perjalanan menuju <i>base</i> dan mengamankan <i>diamond</i> di <i>inventory</i>.</p>
6.	 <p>Kondisi awal</p>	<p>Penanganan red diamond jika sudah menampung 4 diamond: Pada gambar tersebut, kondisi <i>inventory</i> bot sudah berjumlah 4 <i>diamond</i>. Secara natural, bot tidak akan bisa lagi mengambil red diamond yang memiliki bobot 2. Hal ini merupakan implementasi bahwa bot tidak akan mengambil <i>diamond</i> dengan poin 2 ketika total <i>diamond</i> di <i>inventory</i> adalah 4, menghindari <i>error</i> terhadap pengambilan red diamond yang gagal dan menyebabkan bot</p>

	 <p>Kondisi tengah</p>	dalam kondisi <i>invalid move</i> diam di tempat (0, 0).
	 <p>Kondisi akhir</p>	
7.	 <p>Kondisi awal</p>	Menghindari Teleporter: Berdasarkan algoritma yang telah dibuat, bot akan langsung menghindari teleporter jika hal itu menghalangi langkah menuju posisi tujuan. Dalam gambar di samping, terlihat portal yang menghalangi bot untuk mencapai base dalam dua langkah langsung. Potensi bahaya "terlempar" ke posisi yang jauh membuat bot memilih langkah "memutar" untuk menghindari teleport tersebut. Hal ini membuktikan bahwa salah satu strategi optimasi, yaitu "menghindari teleport jika itu dapat merugikan," telah diimplementasikan dengan baik dan sesuai dalam algoritma bot.

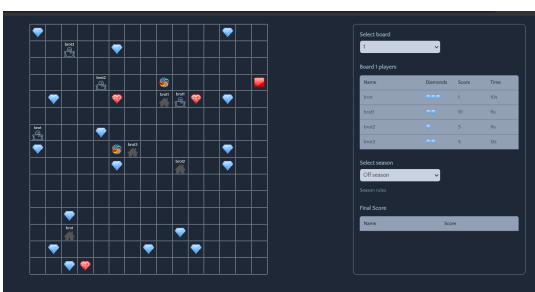


Kondisi tengah



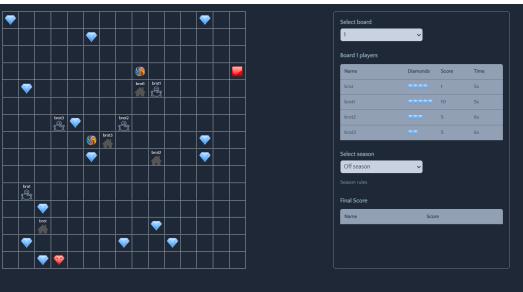
Kondisi akhir

8.

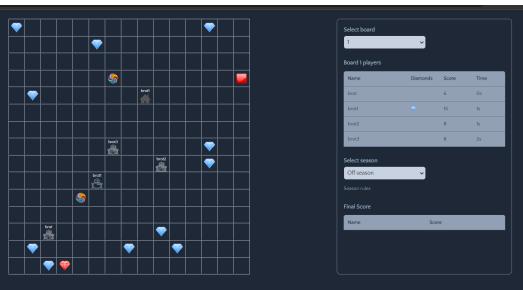


Awal

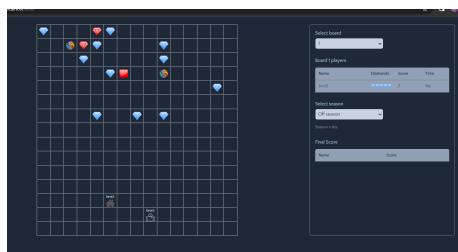
Sekaligus mencari diamond ketika kembali: Perhatikan bort3 pada kasus di samping. Waktu bot ini pada board hanya tersisa 12 detik. Bot tidak akan sempat menyetor diamond pada inventorynya jika dia tetap mencari diamond. Bot ini pun diprogram untuk kembali ke base. Namun, terdapat juga diamond yang dapat diambil bot selagi dia kembali ke base tanpa memerlukan jumlah gerakan tambahan. Jadi, bot tetap dapat mengambil diamond diamond tersebut dan



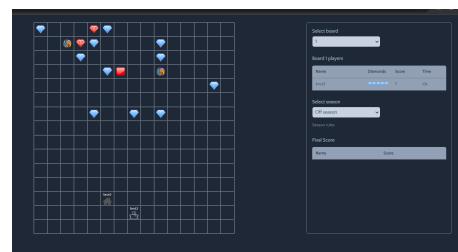
Tengah



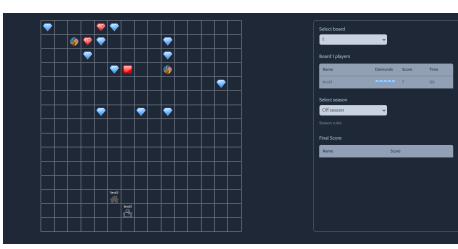
Akhir



Kondisi 1



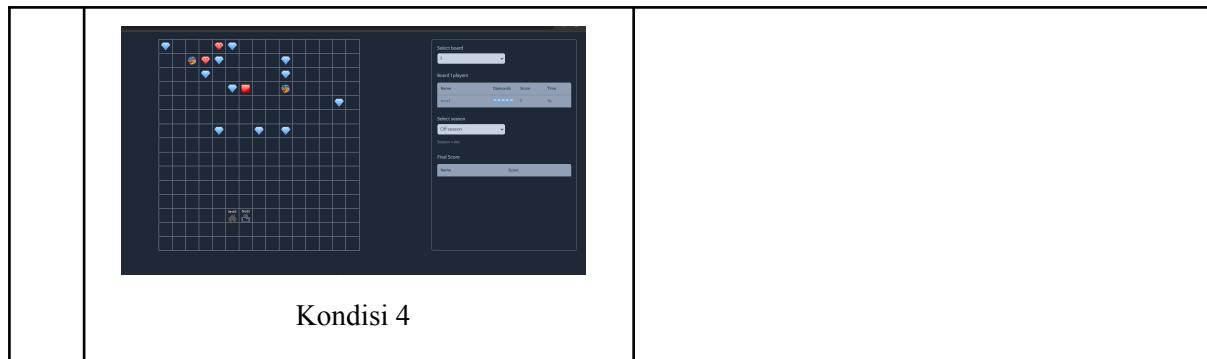
Kondisi 2



Kondisi 3

kembali ke base dengan tepat waktu. Hal ini telah menunjukkan bahwa kami mengimplementasikan penanganan kasus ini.

Gerak secara zig-zag : Kami mengimplementasikan gerakan zig-zag pada bot kami. Bot pada awalnya akan bergerak untuk menyamakan delta x dan delta y antara posisi sekarang dan posisi tujuan bot. Jadi, jika delta x lebih besar dari delta y, bot akan bergerak ke delta x terlebih dahulu. Ketika delta x dan delta y sudah sama, maka bot akan bergerak secara zig-zag yang dimulai dari gerakan ke sumbu y terlebih dahulu. Hal ini melindungi bot kami dari serangan bot lain yang mengasumsikan bahwa kami akan menggunakan gerakan yang disediakan oleh utils.py. Gerakan zig-zag ini juga mengurangi kemungkinan bot harus mengelilingi portal yang menghalangi jalannya.



BAB 5

KESIMPULAN

5.1 Kesimpulan

Pada tugas besar I IF2211 Strategi Algoritma, kami berhasil mengimplementasikan algoritma greedy dalam menyelesaikan permainan Diamonds dengan fokus pada strategi greedy berdasarkan jarak per move terbesar. Algoritma ini terbukti efektif setelah melalui pembahasan, analisis, dan pengujian program, dibandingkan dengan berbagai alternatif algoritma *greedy* lainnya.

Selain memenuhi objektif awal untuk mengumpulkan *diamond* sebanyak mungkin dan menghindari konflik dengan lawan, algoritma yang kami rancang juga mempertimbangkan berbagai kasus yang mungkin terjadi. Hal ini didukung oleh strategi tambahan, seperti penggunaan teleport, memperhitungkan bobot red button, pola *zig-zag* untuk menghindari lawan, pertimbangan sisa waktu sebelum permainan berakhir, kembalinya bot ke base jika jalur yang dilalui tidak mengurangi total move ke goal position, perhitungan inventory, dan menghindari penggunaan teleport jika berpotensi menyebabkan kerugian.

Dengan demikian, algoritma yang kami implementasikan menggabungkan pendekatan algoritma greedy dengan strategi tambahan yang matang untuk menghasilkan performa yang optimal dalam permainan *Diamonds*.

5.2 Saran

Kami merasa program yang kami selesaikan masih jauh dari kata sempurna. Untuk kedepannya, masih banyak pengembangan yang dapat dilakukan untuk memaksimalkan kebermanfaatan program ini. Saran yang dapat diberikan dalam proses pengembangan bot dalam rangka pemenuhan tugas besar I IF2211 Strategi Algoritma ini adalah sebagai berikut.

1. Mengadakan simulasi pertandingan secara rutin dengan berbagai alternatif greedy yang tersedia. Hal ini memungkinkan untuk mengevaluasi kinerja bot dalam berbagai skenario permainan dan memperoleh pemahaman yang lebih baik tentang kelebihan dan kekurangan dari masing-masing strategi yang digunakan.
2. Menyusun implementasi kode dalam struktur yang modular. Dengan memisahkan fungsionalitas-fungsionalitas tertentu ke dalam modul-modul terpisah, proses debugging akan menjadi lebih mudah dilakukan. Ini memungkinkan untuk mengisolasi masalah dan memperbaikinya tanpa mempengaruhi bagian lain dari kode, serta memfasilitasi pengujian unit yang efisien.
3. Adanya perencanaan kelompok yang matang. Dengan merencanakan setiap langkah dengan cermat sebelum melangkah, kelompok dapat memiliki arah yang jelas dalam proses

pembuatan tugas besar. Ini akan membantu menghindari kebingungan, mengurangi risiko kesalahan, dan meningkatkan efisiensi dalam mencapai tujuan bersama.

LAMPIRAN

Tautan Repository GitHub : https://github.com/slntkllr01/Tubes1_tipis_tipis

Tautan Video Youtube : <https://youtu.be/quT3F1LKjg8?si=TufmHWSeiCzJlmnB>

DAFTAR PUSTAKA

- [1] R. Munir, Algoritma Greedy (Bagian 1) - [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)-Bag1.pdf)
- [2] R. Munir, Algoritma Greedy (Bagian 2) - [informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf)
- [3] R. Munir, Algoritma Greedy (Bagian 3) - [informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf)