

**LAPORAN TUGAS BESAR I
IF2211 STRATEGI ALGORITMA**

**PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT
PERMAINAN DIAMONDS**



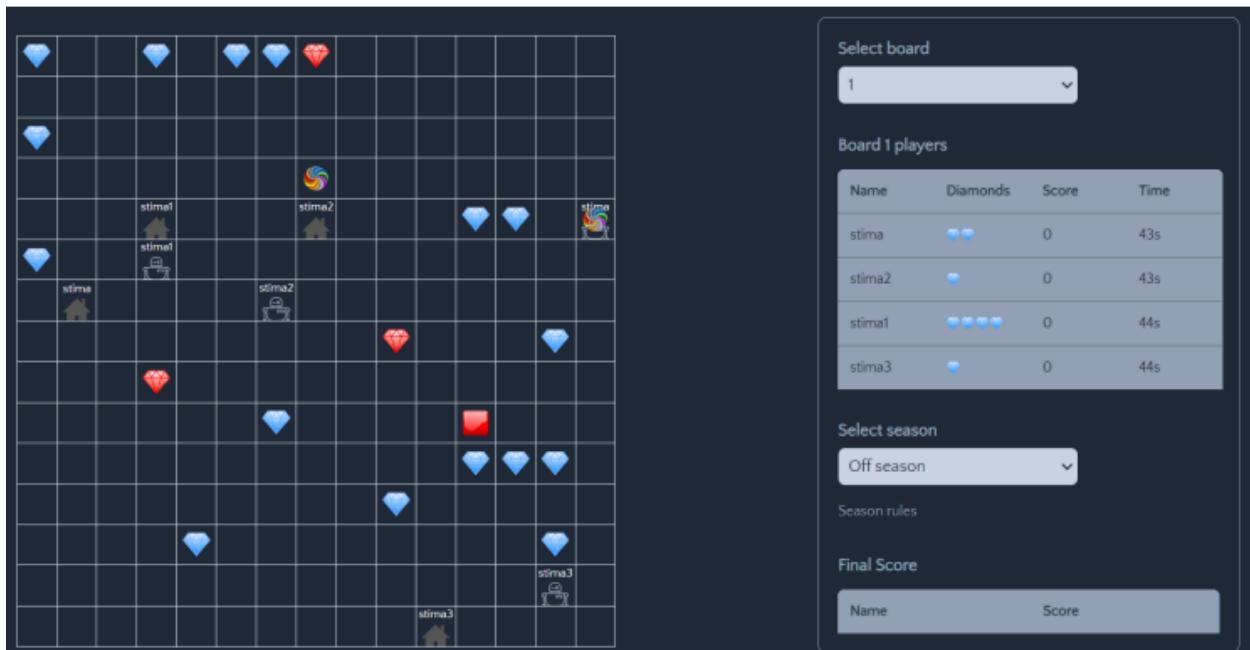
Kelompok TheYuds
Anggota
13522045 Elbert Chailes
13522115 Derwin Rustanly
10023634 Yudi Kurniawan

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

BAB I

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. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.



Gambar 1.1. Tampilan permainan Diamonds yang telah dijalankan

Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi *greedy* dalam membuat bot ini. Program permainan *Diamonds* terdiri atas:

1. Game engine, yang secara umum berisi:

- 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

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan game engine dan membuat bot dari bot starter pack 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*

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

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.

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 II

LANDASAN TEORI

2.1. Algoritma *Greedy*

Algoritma *greedy* adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) sehingga pada setiap langkah mengambil pilihan yang terbaik dan diperoleh pada saat itu tanpa mempertimbangkan konsekuensi kedepannya (prinsip *take what you can get now*) dan berharap bahwa langkah-langkah optimum yang diambil pada kondisi tersebut akan berakhir menjadi langkah paling optimum secara global atau secara hasil akhir.

Terdapat elemen-elemen dalam algoritma *greedy* sebagai berikut.

- a. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah.
- b. Himpunan solusi, S : berisi kandidat yang sudah dipilih
- c. Fungsi solusi : menentukan apakah himpunan yang dipilih sudah memberikan solusi
- d. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
- e. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih layak atau tidak untuk dimasukkan ke dalam himpunan solusi.
- f. Fungsi obyektif: untuk memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen tersebut, maka dapat dipastikan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian S, dari himpunan kandidat C. Maka dari itu, S harus memenuhi beberapa kriteria yang telah ditentukan melalui pemeriksaan fungsi kelayakan, yaitu S menyatakan suatu solusi dan S dioptimasi dengan menggunakan fungsi objektif.

2.2. Game Engine *Diamonds*

Untuk memulai menjalankan permainan dan melakukan pengimplementasian robot dalam permainan, diperlukan *starter pack* permainan *Diamonds*. *Game engine* ini bertujuan untuk menjalankan peta dan media bagi robot untuk bertarung, yang telah dibungkus dalam *docker*, sehingga hanya perlu dilakukan pengaktifan *virtual environment* yang terdapat pada *docker* untuk menjalankan mesin dari permainan *Diamonds*. Permainan *Diamonds* ini merupakan aplikasi yang berbasis *website* sehingga perlu dijalankan tampilan depan *Frontend* dan juga logika di balik permainan tersebut yang dibungkus ke dalam *backend* yang terdapat pada *bot starter pack*. Untuk memulai dan dapat menjalankan game dan mengimplementasikan penggunaan bot diperlukan starter pack game Diamonds. User harus mengunduh starter pack dari game Diamonds. Starter pack game Diamonds dapat diunduh melalui link berikut ini: <https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>.

Ada beberapa komponen penting yang perlu diketahui untuk memahami cara kerja game, yakni:

1. Game Engine: Ini adalah inti dari game Diamonds yang mengatur semua proses di dalamnya. Game engine mengontrol logika permainan, fisika, tampilan grafis, dan interaksi antara pemain dan elemen-elemen permainan lainnya.
2. Backend: Backend merupakan bagian dari game engine yang bertanggung jawab untuk mengelola data dan logika permainan di sisi server. Ini termasuk pengelolaan basis data, pemrosesan perintah pemain, serta logika permainan yang tidak terkait langsung dengan tampilan grafis.
3. Frontend: Frontend adalah bagian dari game engine yang berinteraksi langsung dengan pemain. Ini termasuk tampilan grafis, antarmuka pengguna, dan elemen-elemen visual lainnya yang diperlukan untuk memungkinkan pemain berinteraksi dengan permainan.
4. Karakter dan Objek: Karakter dan objek merupakan entitas dalam permainan yang dapat bergerak dan berinteraksi satu sama lain. Dalam Diamonds, ini mungkin termasuk karakter robot, diamond, hambatan, dan elemen-elemen lain yang dapat mempengaruhi jalannya permainan.

5. Algoritma: Algoritma adalah serangkaian instruksi atau aturan yang digunakan oleh game engine dan bot untuk membuat keputusan dan mengatur perilaku dalam permainan. Algoritma dapat mencakup logika permainan, fisika, kecerdasan buatan, dan strategi permainan lainnya.
6. Paket Starter: Paket starter merupakan kumpulan perangkat lunak dan instruksi yang diperlukan untuk memulai pengembangan atau penggunaan bot dalam permainan. Ini mungkin mencakup kode sumber, dokumen, dan alat-alat lain yang diperlukan untuk mengaktifkan dan mengelola bot dalam permainan Diamonds.

Berikut adalah garis besar cara kerja program game Diamonds:

1. Inisialisasi Pertandingan: Saat engine permainan dijalankan, pertandingan Diamonds akan dimulai dengan engine meng-host pertandingan pada sebuah hostname yang telah ditentukan. Jika berada di lingkungan lokal, engine akan di-host pada localhost:8082.
2. Koneksi antara Engine dan Bot: Setelah engine dijalankan, engine akan mulai melakukan koneksi dengan bot-bot pemain. Engine akan menunggu hingga semua bot pemain terkoneksi. Proses logging juga akan dilakukan dengan cara yang serupa.
3. Persiapan Jumlah Bot: Jumlah bot yang akan bermain dalam satu pertandingan diatur oleh atribut BotCount yang ada dalam file "appsettings.json" di dalam folder "engine-publish".
4. Memulai Pertandingan: Begitu semua bot pemain terkoneksi dan jumlahnya sesuai dengan BotCount yang ditetapkan, pertandingan akan dimulai.
5. Interaksi antara Bot dan Engine: Bot-bot yang terkoneksi akan mendengarkan event-event yang dikirimkan oleh engine. Setiap bot juga akan mengirimkan event kepada engine yang berisi aksi yang akan dilakukan oleh bot.
6. Pelaksanaan Pertandingan: Pertandingan akan berlangsung sampai kondisi pertandingan selesai, seperti mencapai batas waktu atau kondisi kemenangan tertentu. Setelah pertandingan selesai, log akan dicatat dalam file JSON untuk keperluan dokumentasi dan analisis lebih lanjut.

Dengan cara kerja seperti ini, program Diamonds mengatur interaksi antara engine dan bot pemain untuk menyelenggarakan pertandingan Diamonds dan mencatat hasilnya.

2.3. Alur Menjalankan Game *Diamonds*

Berikut adalah langkah-langkah untuk menjalankan program:

1. Jalankan game engine dengan cara mengunduh starter pack game engine dalam bentuk file .zip yang terdapat pada tautan yang telah dilampirkan pada bagian 2.2

- a. Setelah melakukan instalasi, lakukan ekstraksi file .zip tersebut lalu masuk ke root folder dari hasil ekstraksi file tersebut kemudian jalankan terminal
- b. Jalankan perintah berikut pada terminal untuk masuk ke root directory dari game engine

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

- c. Lakukan instalasi dependencies dengan menggunakan yarn. Apabila pengguna belum pernah mengunduh yarn dapat dilakukan instalasi dengan cara melakukan perintah berikut.

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

- d. Lakukan setup environment variable dengan menjalankan script berikut untuk OS Windows

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

Untuk Linux / (possibly) macOS

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

- e. Lakukan setup local database dengan membuka aplikasi docker desktop terlebih dahulu kemudian jalankan perintah berikut di terminal

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

Kemudian 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
```

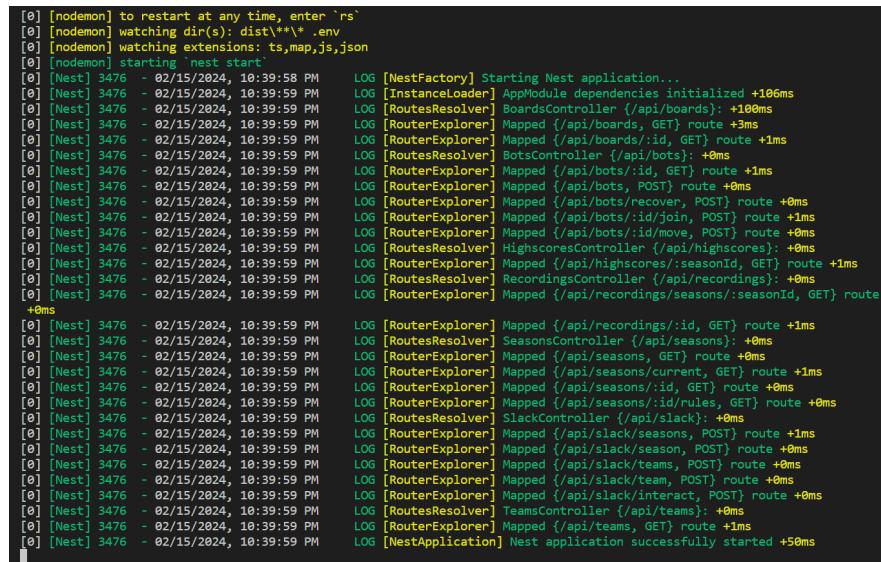
- f. Jalankan perintah berikut untuk melakukan build frontend dari game-engine

```
npm run build
```

- g. Jalankan perintah berikut untuk memulai game-engine

```
npm run start
```

- h. Jika berhasil, tampilan terminal akan terlihat seperti gambar di bawah ini.



```
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching dir(s): dist/**\*.env
[0] [nodemon] watching extensions: ts, map, js, json
[0] [nodemon] starting nest start
[0] [Nest] 3476 - 02/15/2024, 10:39:58 PM  LOG [NestFactory] Starting Nest application...
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [InstanceLoader] AppModule dependencies initialized +106ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] BoardsController {/api/boards}: +100ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/boards, GET} route +3ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/boards/:id, GET} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] BotsController {/api/bots}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/bots/:id, GET} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/bots, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/bots/recover, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/bots/:id/join, POST} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/bots/:id/move, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] HighscoresController {/api/highscores}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/highscores/:seasonId, GET} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] RecordingsController {/api/recordings}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/recordings/seasons/:seasonId, GET} route
+0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/recordings/:id, GET} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] SeasonsController {/api/seasons}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/seasons, GET} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/seasons/current, GET} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/seasons/:id, GET} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/seasons/:id/rules, GET} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] SlackController {/api/slack}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/slack/seasons, POST} route +1ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/slack/season, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/slack/teams, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/slack/team, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/slack/interact, POST} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RoutesResolver] TeamsController {/api/teams}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM  LOG [RouterExplorer] Mapped {/api/teams, GET} route +1ms
[0] [NestApplication] Nest application successfully started +50ms
```

2. Jalankan bot starter pack dengan cara mengunduh kit dengan ekstensi .zip yang terdapat pada tautan yang telah dilampirkan pada bab 1.
 - a. Lakukan ekstraksi file zip tersebut, kemudian masuk ke folder hasil ekstrak tersebut dan buka terminal

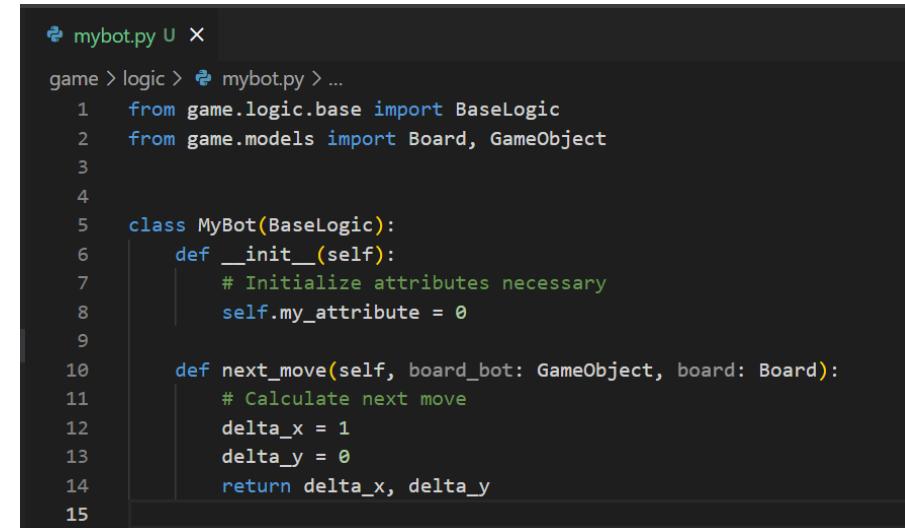
- b. Jalankan perintah berikut untuk masuk ke root directory dari project

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

- c. Jalankan perintah berikut untuk menginstall dependencies dengan menggunakan pip

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

- d. Buatlah folder baru pada direktori /game/logic (misalnya mybot.py)
- e. Buatlah kelas yang meng-inherit kelas *BaseLogic*, lalu implementasikan constructor dan method `next_move` pada kelas tersebut



```
mybotpy U X
game > logic > mybot.py > ...
1  from game.logic.base import BaseLogic
2  from game.models import Board, GameObject
3
4
5  class MyBot(BaseLogic):
6      def __init__(self):
7          # Initialize attributes necessary
8          self.my_attribute = 0
9
10     def next_move(self, board_bot: GameObject, board: Board):
11         # Calculate next move
12         delta_x = 1
13         delta_y = 0
14         return delta_x, delta_y
15
```

- f. Import kelas yang telah dibuat pada main.py dan daftarkan pada dictionary *CONTROLLERS*

```

main.py M X
main.py > ...
1 import argparse
2 from time import sleep
3
4 from colorama import Back, Fore, Style, init
5 from game.api import Api
6 from game.board_handler import BoardHandler
7 from game.bot_handler import BotHandler
8 from game.logic.random import RandomLogic
9 from game.util import *
10 from game.logic.base import BaseLogic
11 | from game.logic.mybot import MyBot ←
12
13 init()
14 BASE_URL = "http://localhost:3000/api"
15 DEFAULT_BOARD_ID = 1
16 CONTROLLERS = {"Random": RandomLogic, "MyBot": MyBot}
17

```

- g. Jalankan program dengan cara menjalankan perintah berikuts.

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

Anda juga bisa menjalankan satu bot saja atau beberapa bot menggunakan .bat atau .sh script.

- Untuk windows

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

- Untuk Linux / (possibly) macOS

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

BAB III

APLIKASI STRATEGI GREEDY

3.1. Alternatif *Greedy*

3.1.1. Alternatif *Greedy by Shortest Displacement*

Greedy by Shortest Displacement adalah strategi *greedy* yang mengutamakan perpindahan terdekat dari bot menuju *diamond*. *Diamond* yang tidak dipedulikan merupakan warna biru atau merah, karena bot hanya mempertimbangkan *diamond* yang berada dengan perpindahan terdekat darinya. Untuk menentukan jarak terdekat antara bot dengan *diamond*, juga dipertimbangkan perpindahan *bot* ke *diamond* dengan menggunakan portal.

1. Pemetaan Elemen Greedy

- a. Himpunan Kandidat: Himpunan *diamond* yang tersedia pada *board*, yang meliputi *diamond* merah yang bernilai 2 poin dan *diamond* biru yang bernilai 1 poin.
- b. Himpunan Solusi: Himpunan *diamond* yang terpilih.
- c. Fungsi Solusi: Menjumlahkan seluruh poin *diamond* yang telah didapatkan.
- d. Fungsi Seleksi: Memilih *diamond* yang berjarak terdekat dari bot.
- e. Fungsi kelayakan: Memeriksa jumlah *diamond* yang telah diperoleh tidak melebihi 5 pada setiap iterasi, dan apabila jumlah *diamond* telah mencapai 5, bot akan diarahkan untuk kembali ke base untuk menyimpan *diamond*.
- f. Fungsi obyektif: Jumlah poin *diamond* yang diperoleh maksimum.

2. Analisis Efisiensi Solusi

Pada alternatif ini, akan dilakukan inisialisasi terhadap suatu *array* kosong yang nantinya menampung keseluruhan jarak dari keseluruhan *diamond* yang tersedia pada *board*. Kemudian, dilakukan iterasi untuk setiap *diamond* yang tersedia pada *board* untuk memasukkan jarak dari *diamond* tersebut pada *array*. Setelah *array* terdefinisi, akan dilakukan pemilihan *diamond* dengan

mempertimbangkan jarak minimum *diamond* tersebut dari koordinat posisi bot pada waktu yang bersangkutan. Kompleksitas algoritma untuk menentukan jarak minimum dari suatu array adalah $O(n)$ dan penentuan jarak tersebut akan dilakukan untuk setiap objek sejumlah n buah yang akan diambil oleh bot, sehingga kompleksitas dari keseluruhan algoritmanya adalah $O(n^2)$. Alternatif ini dapat dioptimasi lebih lanjut dengan mempertimbangkan untuk melakukan pengurutan terhadap array berdasarkan jarak secara terurut membesar $O(n \log(n))$ sehingga waktu yang diperlukan untuk menentukan jarak minimumnya adalah $O(1)$, dan kompleksitas dari keseluruhan algoritmanya adalah $O(n \log(n))$.

3. Analisis Efektivitas Solusi

Strategi ini mengutamakan perpindahan terdekat (*shortest displacement*) *bot* menuju *diamond*.

Strategi ini menjadi efektif apabila:

- a. Bot tidak melakukan *spawn* pertama di dekat *bot* lainnya, sehingga tidak rentan untuk ditabrak oleh bot lainnya.
- b. Bot melakukan *spawn* pada daerah yang memiliki konsentrasi *diamond* yang tinggi karena hanya mempertimbangkan *shortest displacement*, maka *bot* dapat mengambil *diamond* dengan cepat dan kembali ke *base* dengan cepat pula.
- c. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* sehingga *bot* dapat menuju ke daerah yang *mengandung* diamond lebih cepat.
- d. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* ketika bot hendak kembali ke *base*.

Strategi ini menjadi tidak efektif apabila:

- a. Jarak *bot* dengan *bot* lawan sangatlah dekat sehingga menyebabkan terjadi *collision* / tabrakan.
- b. Terdapat portal yang menghalangi jalan *bot* sehingga terdapat dua gerakan tambahan yang harus dilakukan untuk kembali ke jalur pengambilan *diamond*.

3.1.2. Alternatif *Greedy by Highest Value*

Greedy by Highest Value adalah strategi *greedy* yang mengutamakan jumlah poin dari *diamond* yang hendak dituju tanpa mempertimbangkan jarak. Jadi, dalam strategi *greedy* ini, *bot* akan menuju cenderung ke *diamond* berwarna merah dibandingkan *diamond* berwarna biru. Namun, jika terdapat kasus *diamond* berwarna merah tidak terdapat pada papan (dengan kata lain, *diamond* merah tidak *spawn* atau sudah habis), maka *bot* didesain untuk mengambil *diamond* berwarna biru secara acak tanpa mempertimbangkan jarak terdekat.

1. Pemetaan Elemen Greedy

- a. Himpunan Kandidat: Himpunan *diamond* yang tersedia pada *board*, yang meliputi *diamond* merah yang bernilai 2 poin dan *diamond* biru yang bernilai 1 poin.
- b. Himpunan Solusi: Himpunan *diamond* yang terpilih.
- c. Fungsi Solusi: Menjumlahkan seluruh poin *diamond* yang telah didapatkan.
- d. Fungsi Seleksi: Memilih *diamond* yang bernilai maksimum pada *board* tanpa mempertimbangkan jarak.
- e. Fungsi kelayakan: Memeriksa jumlah *diamond* yang telah diperoleh tidak melebihi 5 pada setiap iterasi, dan apabila jumlah *diamond* telah mencapai 5, *bot* akan diarahkan untuk kembali ke base untuk menyimpan *diamond*.
- f. Fungsi obyektif: Jumlah poin *diamond* yang diperoleh maksimum.

2. Analisis Efisiensi Solusi

Pada alternatif ini, akan dilakukan inisialisasi terhadap suatu *array* kosong yang nantinya menampung keseluruhan poin yang diperoleh dari keseluruhan *diamond* yang tersedia pada *board*. Kemudian, dilakukan iterasi untuk setiap *diamond* yang tersedia pada *board* untuk memasukkan jumlah poin dari *diamond* tersebut pada *array*. Setelah *array* terdefinisi, akan dilakukan pemilihan *diamond* dengan jumlah poin terbesar tanpa mempertimbangkan jarak minimum *diamond* tersebut dari koordinat posisi *bot* pada waktu yang bersangkutan. Kompleksitas algoritma untuk menentukan *diamond* dengan poin terbesar dari suatu *array*

adalah $O(n)$ dan penentuan poin *diamond* tersebut akan dilakukan untuk setiap objek sejumlah n buah yang akan diambil oleh bot, sehingga kompleksitas dari keseluruhan algoritmanya adalah $O(n^2)$. Alternatif ini dapat dioptimasi lebih lanjut dengan mempertimbangkan untuk melakukan pengurutan terhadap array berdasarkan poinnya secara terurut mengecil $O(n \log(n))$ sehingga waktu yang diperlukan untuk menentukan poin terbesarnya adalah $O(1)$, dan kompleksitas dari keseluruhan algoritmanya adalah $O(n \log(n))$.

3. Analisis Efektivitas Solusi

Strategi ini mengutamakan *bot* untuk bergerak menuju *diamond* yang menawarkan poin yang lebih tinggi, dengan kata lain memprioritaskan *diamond* berwarna merah.

Strategi ini menjadi efektif apabila:

- a. Bot melakukan *spawn* pada area yang memiliki konsentrasi *diamond* berwarna merah yang tinggi
- b. Ketika papan permainan *Diamonds*, memiliki spawn *diamond* berwarna merah yang ditandai dengan jumlah pemain yang banyak.

Strategi ini menjadi tidak efektif apabila:

- a. *Diamond* berwarna merah yang *spawn* di papan permainan cukup sedikit sehingga poin potensial yang dapat diperoleh *bot* juga menjadi lebih rendah.
- b. Jika *diamond* berwarna merah sudah habis diambil oleh *bot* lawan sehingga *bot* dengan algoritma akan cenderung mengambil secara acak *diamond* berwarna biru yang terdapat pada papan permainan.

3.1.3. Alternatif *Greedy by Highest Density*

Greedy by Highest Density adalah strategi *greedy* yang mengutamakan densitas tertinggi dari hasil pembagian poin yang didapatkan dari sebuah *diamond* dibagikan dengan perpindahan yang harus ditempuh untuk mendapatkan *diamond* tersebut. Dalam strategi *greedy* ini, *bot* juga mempertimbangkan penggunaan

portal. Jika dalam penggunaan portal, *bot* menemukan *highest density* yang lebih tinggi maka *bot* akan menuju *diamond* dengan menggunakan *portal*.

1. Pemetaan Elemen Greedy

- a. Himpunan Kandidat: Himpunan *diamond* yang tersedia pada *board*, yang meliputi *diamond* merah yang bernilai 2 poin dan *diamond* biru yang bernilai 1 poin.
- b. Himpunan Solusi: Himpunan *diamond* yang terpilih.
- c. Fungsi Solusi: Menjumlahkan seluruh poin *diamond* yang telah didapatkan.
- d. Fungsi Seleksi: Memilih *diamond* dengan densitas yang terbesar dari *diamond* lainnya.
- e. Fungsi kelayakan: Memeriksa jumlah *diamond* yang telah diperoleh tidak melebihi 5 pada setiap iterasi, dan apabila jumlah *diamond* telah mencapai 5, *bot* akan diarahkan untuk kembali ke base untuk menyimpan *diamond*.
- f. Fungsi obyektif: Jumlah poin *diamond* yang diperoleh maksimum.

2. Analisis Efisiensi Solusi

Pada alternatif ini, akan dilakukan inisialisasi terhadap suatu *array* kosong yang nantinya menampung keseluruhan densitas poin terhadap jarak yang diperoleh dari keseluruhan *diamond* yang tersedia pada *board*. Kemudian, dilakukan iterasi untuk setiap *diamond* yang tersedia pada *board* untuk memasukkan jumlah densitas poin terhadap jarak dari *diamond* tersebut pada *array*. Setelah *array* terdefinisi, akan dilakukan pemilihan *diamond* dengan jumlah densitas poin yang terbesar. Alternatif solusi ini dapat terbilang lebih efisien apabila disandingkan dengan alternatif *greedy by highest value* sebelumnya karena juga mempertimbangkan faktor jarak terdekat sekaligus poin terbesar dari *diamond* yang akan dipilih. Kompleksitas algoritma untuk menentukan *diamond* dengan densitas poin terbesar dari suatu *array* adalah $O(n)$ dan penentuan densitas poin *diamond* tersebut akan dilakukan untuk setiap objek sejumlah n buah yang akan diambil oleh *bot*, sehingga kompleksitas dari keseluruhan algoritmanya adalah $O(n^2)$. Alternatif ini dapat dioptimasi lebih lanjut dengan mempertimbangkan untuk melakukan pengurutan terhadap *array*

berdasarkan densitas poinnya secara terurut mengecil $O(n \log(n))$ sehingga waktu yang diperlukan untuk menentukan densitas poin terbesarnya adalah $O(1)$, dan kompleksitas dari keseluruhan algoritmanya adalah $O(n \log(n))$.

3. Analisis Efektivitas Solusi

Konsep dalam strategi *greedy* ini mirip dengan alternatif *greedy by shortest displacement*, namun hanya terdapat perbedaan dalam parameter fungsi seleksi yang mana alternatif ini mempertimbangkan densitas, sedangkan alternatif yang satunya lagi mempertimbangkan perpindahan.

Strategi ini menjadi efektif apabila:

- a. Bot tidak melakukan *spawn* pertama di dekat *bot* lainnya, sehingga tidak rentan untuk ditabrak oleh bot lainnya.
- b. Bot melakukan *spawn* pada daerah yang memiliki konsentrasi *diamond* yang tinggi karena hanya mempertimbangkan *shortest displacement*, maka *bot* dapat mengambil *diamond* dengan cepat dan kembali ke *base* dengan cepat pula.
- c. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* sehingga *bot* dapat menuju ke daerah yang *mengandung* diamond lebih cepat.
- d. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* ketika bot hendak kembali ke *base*.
- e. Terdapat *bot* yang mengejarnya atau *bot* yang terletak satu blok darinya sehingga *bot* akan *ter-trigger* untuk melakukan tabrakan dengan *bot* lawan.

Strategi ini menjadi tidak efektif apabila:

- a. Jarak *bot* dengan *bot* lawan sangatlah dekat sehingga menyebabkan terjadi *collision* / tabrakan.
- b. Terdapat portal yang menghalangi jalan *bot* sehingga terdapat dua gerakan tambahan yang

- c. Terdapat *diamond* merah dan *diamond* biru dengan perpindahan yang sama dengan *bot*, namun *bot* memilih *diamond* berwarna merah yang berada pada daerah yang berkonsentrasi *diamond* rendah.

3.1.4. Alternatif *Greedy by Nearest Base*

Greedy by Nearest Base adalah strategi *greedy* yang mengutamakan pengambilan *diamond-diamond* yang memiliki densitas yang paling tinggi dengan *base* daripada *bot*. Penerapan densitas pada algoritma ini sama dengan yang diterapkan pada alternatif *greedy by highest density*.

1. Pemetaan Elemen Greedy

- a. Himpunan Kandidat: Himpunan *diamond* yang tersedia pada *board*, yang meliputi *diamond* merah yang bernilai 2 poin dan *diamond* biru yang bernilai 1 poin.
- b. Himpunan Solusi: Himpunan *diamond* yang terpilih.
- c. Fungsi Solusi: Menjumlahkan seluruh poin *diamond* yang telah didapatkan.
- d. Fungsi Seleksi: Memilih *diamond* dengan densitas terbesar dan dengan jarak yang terdekat dari *base*.
- e. Fungsi kelayakan: Memeriksa jumlah *diamond* yang telah diperoleh tidak melebihi 5 pada setiap iterasi, dan apabila jumlah *diamond* telah mencapai 5, *bot* akan diarahkan untuk kembali ke *base* untuk menyimpan *diamond*.
- f. Fungsi obyektif: Jumlah poin *diamond* yang diperoleh maksimum.

2. Analisis Efisiensi Solusi

Pada alternatif ini, diimplementasikan algoritma pengambilan *diamond* yang serupa dengan algoritma *highest density* dengan uraian sebagai berikut. Kompleksitas algoritma untuk menentukan *diamond* dengan densitas poin terbesar dari suatu array adalah $O(n)$ dan penentuan densitas poin *diamond* tersebut akan dilakukan untuk setiap objek sejumlah n buah yang akan diambil oleh *bot*, sehingga kompleksitas dari keseluruhan algoritmanya adalah $O(n^2)$. Alternatif ini dapat dioptimasi lebih lanjut dengan mempertimbangkan untuk melakukan pengurutan terhadap array berdasarkan densitas poinnya secara terurut mengecil $O(n \log(n))$ sehingga waktu yang diperlukan untuk menentukan densitas

poin terbesarnya adalah $O(1)$, dan kompleksitas dari keseluruhan algoritmanya adalah $O(n \log(n))$.

3. Analisis Efektivitas Solusi

Konsep dalam strategi *greedy* ini mirip dengan alternatif *greedy by density*, namun hanya terdapat perbedaan pada prioritas pengambilan *diamond*. *Bot* memiliki fokus untuk mengambil *diamond* yang terletak dengan *base*-nya terlebih dahulu dibandingkan mengambil *diamond* yang berjarak jauh. Namun, jika *bot* tidak menemukan *diamond-diamond* yang cukup, maka ia akan kembali ke alternatif *greedy by highest density*.

Strategi ini menjadi efektif apabila:

- a. Bot tidak melakukan *spawn* pertama di dekat *bot* lainnya, sehingga tidak rentan untuk ditabrak oleh bot lainnya.
- b. Bot melakukan *spawn* pada daerah yang memiliki konsentrasi *diamond* yang tinggi karena hanya mempertimbangkan *shortest displacement*, maka *bot* dapat mengambil *diamond* dengan cepat dan kembali ke *base* dengan cepat pula.
- c. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* sehingga *bot* dapat menuju ke daerah yang mengandung *diamond* lebih cepat.
- d. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* ketika bot hendak kembali ke *base*.
- e. Terdapat banyak *diamond* yang *spawn* di sekitar *base* sehingga *bot* dapat memprioritaskan *diamond* yang berada dekat dengan *base*-nya.

Strategi ini menjadi tidak efektif apabila:

- a. Jarak *bot* dengan *bot* lawan sangatlah dekat sehingga menyebabkan terjadi *collision* / tabrakan.
- b. Terdapat portal yang menghalangi jalan *bot* sehingga terdapat dua gerakan tambahan yang

- c. Terdapat *diamond* merah dan *diamond* biru dengan perpindahan yang sama dengan *bot*, namun *bot* memilih *diamond* berwarna merah yang berada pada daerah yang berkonsentrasi *diamond* rendah.
- d. Konsentrasi *diamond* tinggi yang muncul di sekitar *base*-nya ketika ia sudah jauh dari *base*-nya dan musuh sudah mengambil terlebih dahulu *diamond* yang berada di dekat *base*-nya.

3.1.5. Alternatif *Greedy by Highest Concentration*

Greedy by Highest Concentration adalah strategi *greedy* yang mengutamakan konsentrasi, yakni hasil pembagian jumlah *diamond* pada suatu sektor dengan jarak terdekat *diamond* tersebut dengan posisi *bot*, sehingga bernilai tertinggi. Strategi ini merupakan bentuk optimasi terhadap strategi *greedy by lowest displacement* dengan cara membagi map utama menjadi 4 sektor dan memilih sektor dengan konsentrasi *diamond* terbesar, setelah ditemukan sektor yang konsentrasi *diamond*-nya terbesar. Dalam strategi *greedy* ini, *bot* juga mempertimbangkan penggunaan portal. Jika dalam penggunaan portal, *bot* menemukan konsentrasi sektor yang lebih tinggi maka *bot* akan menuju *diamond* pada sektor dengan menggunakan *portal*.

1. Pemetaan Elemen Greedy
 - a. Himpunan Kandidat: Himpunan *diamond* yang tersedia pada *board*, yang meliputi *diamond* merah yang bernilai 2 poin dan *diamond* biru yang bernilai 1 poin.
 - b. Himpunan Solusi: Himpunan *diamond* yang terpilih.
 - c. Fungsi Solusi: Menjumlahkan seluruh poin *diamond* yang telah didapatkan.
 - d. Fungsi Seleksi: Memilih *diamond* pada sektor yang densitasnya terbesar dari sektor lainnya.
 - e. Fungsi kelayakan: Memeriksa jumlah *diamond* yang telah diperoleh tidak melebihi 5 pada setiap iterasi, dan apabila jumlah *diamond* telah mencapai 5, *bot* akan diarahkan untuk kembali ke *base* untuk menyimpan *diamond*.
 - f. Fungsi obyektif: Jumlah poin *diamond* yang diperoleh maksimum.

2. Analisis Efisiensi Solusi

Pada alternatif ini, diimplementasikan algoritma pengambilan *diamond* yang serupa dengan *greedy by shortest displacement* dengan uraian sebagai berikut. Kompleksitas algoritma untuk melakukan sektorisasi adalah $O(1)$ karena sektorisasi dilakukan secara konstan dan tidak bergantung terhadap banyaknya *diamond* (n). Kompleksitas algoritma untuk menentukan *diamond* dengan jarak minimum terhadap posisi bot dari suatu array adalah $O(n)$ dan penentuan jarak *diamond* tersebut akan dilakukan untuk setiap objek sejumlah n buah yang akan diambil oleh bot, sehingga kompleksitas dari keseluruhan algoritmanya adalah $O(n^2)$. Alternatif ini dapat dioptimasi lebih lanjut dengan mempertimbangkan untuk melakukan pengurutan terhadap array berdasarkan jarak *diamondnya* secara terurut membesar $O(n \log(n))$ sehingga waktu yang diperlukan untuk menentukan densitas poin terbesarnya adalah $O(1)$, dan kompleksitas dari keseluruhan algoritmanya adalah $O(n \log(n))$.

3. Analisis Efektivitas Solusi

Konsep dalam strategi *greedy* ini mirip dengan alternatif *greedy by shortest displacement*, namun hanya terdapat perbedaan dalam parameter fungsi seleksi yang mana alternatif ini mempertimbangkan densitas, sedangkan alternatif yang satunya lagi mempertimbangkan perpindahan.

Strategi ini menjadi efektif apabila:

- a. Bot tidak melakukan *spawn* pertama di dekat *bot* lainnya, sehingga tidak rentan untuk ditabrak oleh bot lainnya.
- b. Bot melakukan *spawn* pada daerah yang dekat dengan daerah yang memiliki konsentrasi *diamond* yang tinggi, sehingga *bot* akan sempat untuk mendahului *bot* lawan dalam pengamanan *diamond*.
- c. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* sehingga *bot* dapat menuju ke daerah yang *mengandung* diamond lebih cepat.
- d. Terdapat portal yang membantu *bot* untuk memperpendek *displacement* ketika bot hendak kembali ke base.

Strategi ini menjadi tidak efektif apabila:

- a. Jarak *bot* dengan *bot* lawan sangatlah dekat sehingga menyebabkan terjadi *collision* / tabrakan.
- b. Terdapat *diamond* merah dan *diamond* biru dengan perpindahan yang sama dengan *bot*, namun *bot* memilih *diamond* berwarna merah yang berada pada daerah yang berkonsentrasi *diamond* rendah.
- c. *Bot* akan memilih daerah dengan konsentrasi *diamond* yang tinggi, namun daerah dengan konsentrasi tinggi tersebut sangat jauh dari *base* sehingga *bot* akan memakan waktu yang lama untuk kembali ke *base*.

3.2. Strategi *Greedy* yang Diimplementasikan

Berdasarkan alternatif-alternatif *greedy* yang telah dijelaskan pada bagian 3.1, dapat diamati bahwa terdapat alternatif *greedy* yang masing-masing memiliki kelebihan dan kekurangannya masing-masing. Namun, pada akhirnya, diputuskan untuk menggunakan salah satu alternatif tersebut, yaitu strategi *greedy by highest density*. Alasan penggunaan hanya salah satu strategi dan tidak melakukan kombinasi ketiga strategi tersebut adalah dengan alasan bahwa “semakin pintar atau semakin banyak berpikir sebuah *bot*, maka ia akan bertindak terlalu ragu-ragu dalam mengambil keputusannya”, yang terbukti dengan hasil tes pertandingan lokal yang dilakukan, bahwa *bot* yang konsisten untuk memperkaya dirinya sendiri dengan mengambil sebanyak-banyaknya *diamond*, terbukti bahwa *greedy by highest density* lebih sering memenangkan permainan.

Alasan alternatif *greedy by shortest displacement* tidak dinyatakan sebagai solusi paling optimal dalam implementasi karena faktor poin belum dipertimbangkan dalam pengambilan keputusan bot dalam menyeleksi *diamond* yang akan diambil.

Alasan alternatif *greedy by highest value* tidak dipertimbangkan dalam penggunaan adalah faktor ketidakkonsistenan performa *bot* dengan algoritma, karena *bot* akan cenderung kehilangan *performance*-nya ketika *diamond* berwarna merah telah habis dari papan permainan sehingga ia akan cenderung memiliki skor yang lebih kecil pada saat pertandingan.

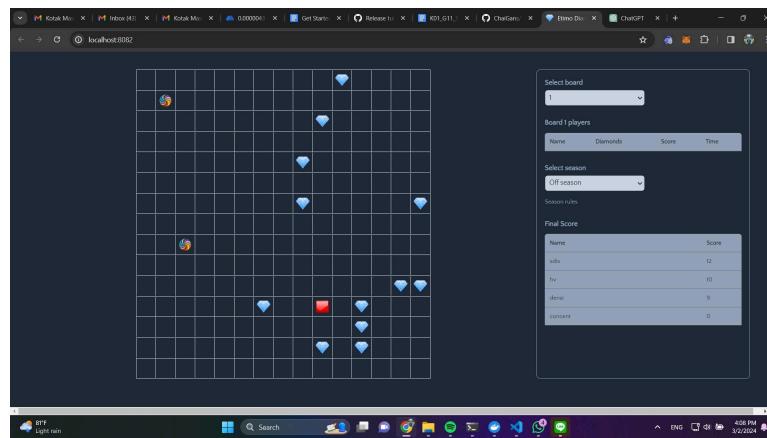
Alasan alternatif *greedy by highest concentration* tidak digunakan karena ketika *bot* terlalu banyak melakukan pemilihan atau seleksi *diamond* yang hendak diambil sehingga menyebabkan ketidakefektifan, seperti yang telah disebut pada analisis efektivitas solusi bahwa

alternatif ini akan tidak efektif ketika daerah yang dikunjungi terlalu jauh karena *diamond* telah diambil oleh *bot* lawan lain ketika *bot* sampai.

Alasan alternatif *greedy by nearest base* tidak digunakan adalah dengan pertimbangan yang hampir sama seperti *highest concentration*, ketika ia sudah berada di posisi yang terlalu jauh dari *base*-nya sendiri, ia akan kembali ke *base*-nya sendiri, namun karena konsentrasi daerah tersebut kian berubah ketika diambil oleh *bot* lain, *bot* akan mempertimbangkan untuk berubah alternatif ke penerapan *highest density*, sehingga inkonsistensi ini menyebabkan ketidak efektivitas alternatif ini.

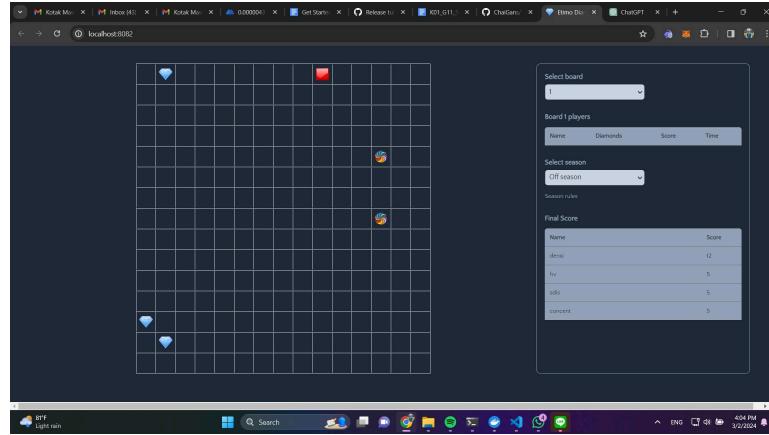
Terdapat juga beberapa kali hasil permainan yang melakukan antar alternatif di atas untuk mendukung analisis yang dilakukan untuk menentukan alternatif mana yang hendak digunakan. Berikut merupakan hasil pertandingan dari alternatif tersebut. Sebagai informasi tambahan, bahwa *bot* sdis menerapkan alternatif *greedy by shortest displacement*, *bot* hv menerapkan alternatif *greedy by highest value*, *bot* densi menerapkan alternatif *greedy by highest density*, dan *bot* concen menerapkan alternatif *greedy by highest concentration*.

1. Laga 1



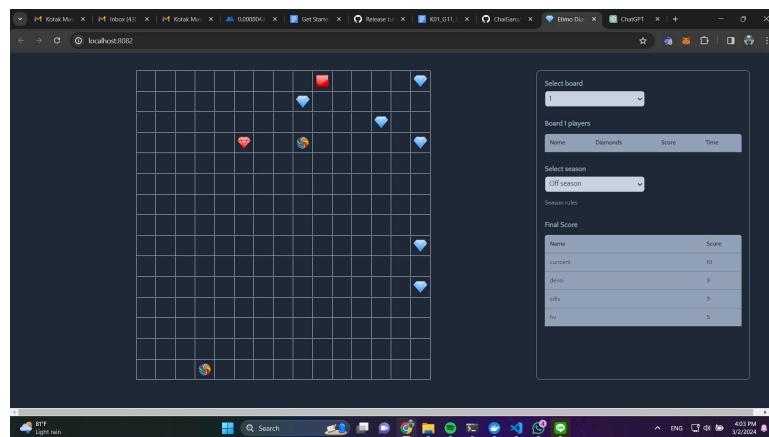
Gambar 3.2.1. Hasil pertandingan 1 antar alternatif

2. Laga 2



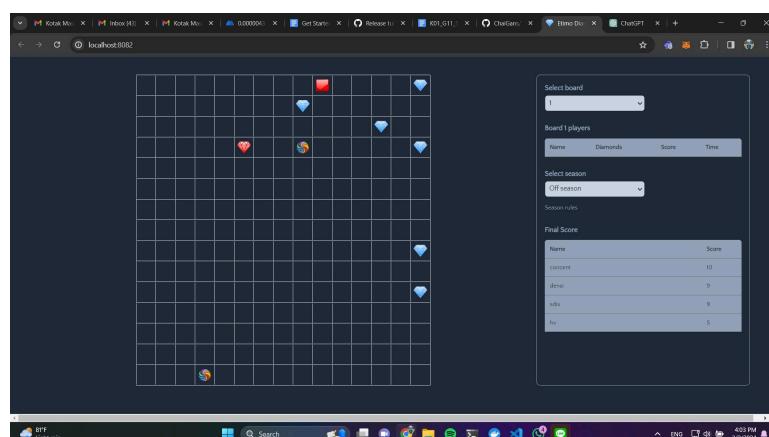
Gambar 3.2.2. Hasil pertandingan 2 antar alternatif

3. Laga 3



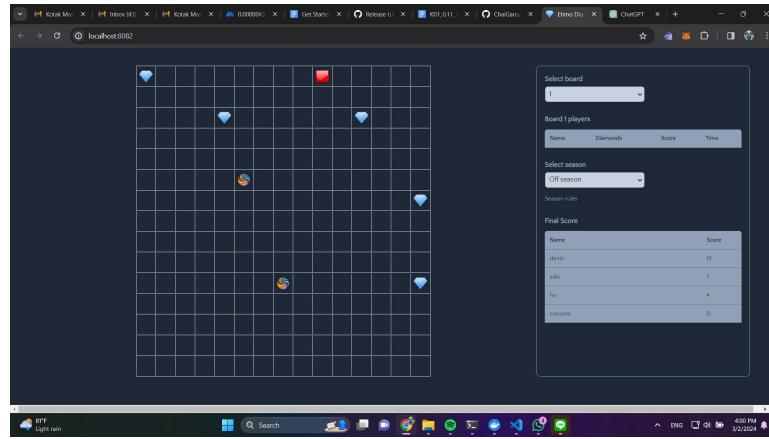
Gambar 3.2.3. Hasil pertandingan 3 antar alternatif

4. Laga 4



Gambar 3.2.4. Hasil pertandingan 4 antar alternatif

5. Laga 5



Gambar 3.2.5. Hasil pertandingan 5 antar alternatif

Dari hasil pertandingan yang dilakukan sebanyak lima kali, terlihat bahwa terjadi perbedaan performa yang ditandai dengan perubahan peringkat yang diperoleh tiap alternatif dalam setiap pertandingannya. Namun, dapat diamati bahwa *bot* densi bisa dibilang medioker, yang mana ia memiliki performa yang lebih konsisten dibandingkan dengan lainnya, yang ditandai juara 1 sebanyak dua kali, juara 2 sebanyak dua kali, dan juara 3 sebanyak sekali. Oleh karena itu, disimpulkan bahwa alternatif paling konsisten, meskipun terdapat peta-peta permainan yang berbeda, namun alternatif tersebut dapat mempertahankan performanya.

Sebenarnya pula, terdapat alternatif seperti *greedy by collision* yang merupakan alternatif *greedy* yang sebenarnya cukup potensial untuk diterapkan dalam permainan, namun *greedy by collision* akan juga memberikan risiko yang besar terhadap performa *bot* karena terdapat *miss rate* yang tinggi pula. Jadi, alternatif *greedy* ini tidak bisa dijadikan strategi utama, namun diterapkan pada *strategi* lainnya sebagai *strategi* pembantu pada kasus-kasus tertentu. Sebagai informasi tambahan, *strategi collision* ini diterapkan ke alternatif *greedy by highest density* untuk menunjang performansi daripada alternatif *greedy* tersebut.

Dengan itu, dapat disimpulkan kembali bahwa alternatif *greedy* yang diimplementasikan adalah alternatif *greedy by highest density*.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi dalam *Pseudocode*

Fungsi displacement

```
{ Fungsi displacement merupakan fungsi untuk menghitung perpindahan dua titik }

function displacement(current_position: Position, goal_position: Position) -> real
    → sqrt((current_position.x - goal_position.x)^2 + (current_position.y - goal_position.y)^2)
```

Fungsi possible_direction

```
{ Fungsi possible direction merupakan fungsi yang mengembalikan list of tuple of integer yang menggambarkan pergerakan yang mungkin diimplementasikan oleh bot pada posisi bot saat itu }

function possible_direction(current_position: Position, board: Board) -> List of Tuple of integer
    direction_available = {}
    if current_position.x > 0 then
        direction_available.append((-1, 0))
    if current_position.x < board.width - 1 then
        direction_available.append((1, 0))

    if current_position.y > 0 then
        direction_available.append((0, -1))
    if current_position.y < board.height - 1 then
        direction_available.append((0, 1))

    → direction_available
```

Fungsi portal_utility_displacement

```
{ Fungsi portal_utility_displacement merupakan fungsi untuk menentukan portal mana yang lebih dekat dengannya dan total perpindahan bot dari posisi awalnya menuju portal inisial dan perpindahan portal pasangannya menuju diamond yang terdekat }
```

```

function portal_utility_displacement(target: String, current_position: Position,
portal_one_position: Position, portal_two_position: Position, board: Board,
board_bot: GameObject) → Tuple real(Position, Float)
    displacement_to_portal_one ← displacement(current_position,
    portal_one_position) { menghitung jarak bot terhadap portal 1 }
    displacement_to_portal_two ← displacement(current_position,
    portal_two_position) { menghitung jarak bot terhadap portal 2 }

    { initial_to_portal_displacement berarti jarak bot menuju portal terdekatnya
    atau portal masuknya }
    { portal_to_finish berarti portal keluar bot }
    { initial_portal berarti portal masuk bot }
    if displacement_to_portal_one ≤ displacement_to_portal_two then
        initial_to_portal_displacement ← displacement_to_portal_one
        portal_to_finish ← portal_two_position
        initial_portal ← portal_one_position
    else
        initial_to_portal_displacement ← displacement_to_portal_two
        portal_to_finish ← portal_one_position
        initial_portal ← portal_two_position

    { Jika bot sedang ingin bergerak menuju diamond }
    { displacement_to_diamond berisi Tuple (Posisi diamond, displacement diamond
    terhadap portal keluar bot) }
    if target == "DiamondGameObject" then
        displacement_to_diamond ← {}
        for diamond in board.diamonds do
            displacement_to_diamond.append((diamond.position,
            displacement(portal_to_finish, diamond.position)))
        shortest_displacement ← min(displacement_to_diamond) depends on
        displacement_to_diamond[1]
        shortest_displacement_to_diamond ← shortest_displacement[1]

        { total_displacement merupakan perpindahan total bot ke portal masuk
        dan bot dari portal keluar menuju diamond terdekat }
        total_displacement ← shortest_displacement_to_diamond +
        initial_to_portal_displacement

    { Jika bot sedang ingin bergerak menuju base }
    if target == "Base" then
        base_position = board_bot.properties.base

```

```

displacement_to_base = displacement(portal_to_finish, base_position)
{ total_displacement merupakan perpindahan total bot ke portal masuk
dan bot dari portal keluar menuju base }
total_displacement = displacement_to_base +
initial_to_portal_displacement

→ initial_portal, total_displacement

```

Fungsi next_move

{ Fungsi next_move merupakan fungsi untuk menentukan langkah selanjutnya yang akan dilalui bot berdasarkan algoritma greedy. Fungsi ini mengembalikan tuple of integer yang merupakan `(delta_x,delta_y)`.

```

function next_move(board_bot: GameObject, board: Board) → Tuple of integer
    { Inisialisasi }

    { Terdapat juga inisialisasi goal_position yang merupakan variabel yang
    terdapat pada class HighestDensity ini , yang diinisiasi pada class dan di
    luar fungsi ini}

    props ← board_bot.properties
    current_position ← board_bot.position
    first_portal_position, second_portal_position ← find_portal_position(board)
    base = board_bot.properties.base
    time_rem ← props.milliseconds_left
    direction_available ← possible_direction(current_position, board)
    enemy_bot ← find_enemy_bots(board)
    diamond_near_base = count_diamond_near_base(board, board_bot)

    { Collision Strategy }
    for enemy in enemy_bot do
        { Alasan untuk melakukan mod 1000 pada milliseconds left adalah untuk
        mengetahui bot mana yang akan melakukan pergerakan terlebih dahulu }
        if (enemy.properties.diamonds > board_bot.properties.diamonds and
        board_bot.properties.diamonds < 3 and
        (board_bot.properties.milliseconds_left mod 1000 <
        enemy.properties.milliseconds_left mod 1000)) then
            if (abs(current_position.x - enemy.position.x) = 1 and
            current_position.y = enemy.position.y) then
                → [enemy.position.x - current_position.x, 0]
            if (abs(current_position.y - enemy.position.y) = 1 and

```

```

        current_position.x = enemy.position.x) then
            → [0, enemy.position.y - current_position.y]

    { Jika bot sedang memegang 5 diamonds, maka bot ditugaskan untuk segera
    kembali ke base }

    if props.diamonds = 5 then
        initial_portal_position, effective_portal_base_displacement ←
        portal_utility_displacement("Base",
        current_position, first_portal_position, second_portal_position, board,
        board_bot)
        if (displacement(current_position,base) >
        effective_portal_base_displacement) then
            goal_position ← initial_portal_position
        else
            goal_position ← base

    {Kasus ketika robot baru masuk ke dalam teleporter dan inventory = 5}
    if (is_teleporter_position(current_position, board)) then
        for direction in direction_available do
            expected_position ←
            Position(current_position.y+direction[1],
            current_position.x+direction[0])
            if (not is_teleporter_position(expected_position, board))
            then
                goal_position ← base
    else
        listRatio ← {}
        for diamond in board.diamonds do
            if props.diamonds = 4 then
                if diamond.properties.points ≠ 2 then
                    listRatio.append((diamond.position,
                    diamond.properties.points/displacement(board_bot.po
                    sition,diamond.position),
                    displacement(board_bot.position,diamond.position)))
                else
                    listRatio.append((diamond.position,
                    diamond.properties.points/displacement(board_bot.position,
                    diamond.position),
                    displacement(board_bot.position,diamond.position)))
            maxDiamond ← max(listRatio) depends on listRatio[1]

```

```

initial_portal_position, effective_portal_diamond_displacement ←
portal_utility_displacement("DiamondGameObject", current_position,
first_portal_position, second_portal_position, board, board_bot)

if (maxDiamond[2] > effective_portal_diamond_displacement) then
    goal_position = initial_portal_position
else
    goal_position = maxDiamond[0]
{Kasus ketika robot baru masuk ke dalam teleporter dan inventory < 5
dan target robot adalah diamond}

if (is_teleporter_position(current_position, board)) then
    for direction in direction_available do
        expected_position ←
        Position(current_position.y+direction[1],
        current_position.x+direction[0])
        if (not is_teleporter_position(expected_position, board))
        then
            if (is_diamond_position(expected_position, board)):
                → direction
            else
                maxDiamond ← max(listRatio, key = lambda x:
                x[1])
            goal_position ← maxDiamond[0]

{Kasus ketika waktu yang tersisa dalam permainan dibawah 10 detik}
if time_rem <= 10000 then
    if(props.diamonds > 1) then
        initial_portal_position,
        effective_portal_base_displacement ←
        portal_utility_displacement("Base",
        current_position, first_portal_position,
        second_portal_position, board, board_bot)
        if (displacement(current_position,base) >
        effective_portal_base_displacement) then
            goal_position = initial_portal_position
        else
            goal_position = base

{Pengembalian nilai fungsi}
delta_x, delta_y = get_direction(
    current_position.x,

```

```

        current_position.y,
        goal_position.x,
        goal_position.y,
    )

    {Error Handling apabila delta_x = 0 dan delta_y = 0}
    if (delta_x = 0 and delta_y = 0) then
        delta_x, delta_y ←
        direction_available[random.randrange(0, len(direction_available))]
    → delta_x, delta_y

```

4.2. Struktur Data Program

Bahasa yang dipilih untuk melakukan implementasi *bot* adalah bahasa pemrograman *Python*. Struktur data pada program didefinisikan dengan menggunakan *class*. *Class* yang digunakan dalam pengembangan bot terdapat pada folder *game/* yang terdapat pada *bot starter pack*.

Class yang terdapat pada *models.py* antara lain :

- *Class bot* merupakan kelas untuk melakukan instansiasi objek *bot*, yang terdiri dari 3 komponen, yaitu nama, email, dan juga id *bot*.
- *Class Position* merupakan kelas yang menampung informasi koordinat x dan y atau cenderung digunakan sebagai tipe data.
- *Class Properties* merupakan kelas yang menampung informasi sebagai berikut.
 - *points* merupakan jumlah poin yang diberikan oleh sebuah *diamond* yang biasanya berpoin 1 atau 2,
 - *pair_id* merupakan informasi yang digunakan untuk melakukan *pairing* pada *Teleporter*,
 - *diamonds* merupakan total *diamond* yang sedang dipegang oleh *bot*,
 - *score* merupakan poin total yang telah disimpan oleh *bot* ke dalam basenya, atau dengan kata lain poin yang ditampilkan pada *scoreboard*,
 - *name* berisi informasi nama dari objek tersebut,
 - *inventory_size* berisi informasi terkait ukuran *inventory* yang dimiliki oleh objeknya, biasanya berukuran 5 untuk objek *bot*,
 - *can_tackle* berisi informasi apakah sebuah objek bisa ditabrak atau tidak,

- *milliseconds_left* berisi informasi terkait waktu yang tersisa oleh sebuah objek *bot* untuk tetap bermain dalam papan permainan,
- *time_joined* berisi informasi waktu masuknya sebuah objek ke dalam permainan,
- *base* berisi informasi posisi letak *base* dari sebuah objek jika ada.
- *Class GameObject* merupakan kelas yang menampung informasi berupa id, posisi, tipe, dan *properties* daripada sebuah objek yang diinstansiasi dengan menggunakan *blueprint* kelas ini. *Properties* merupakan *extension class* dari kelas yang sebelumnya.
- *Class Board* merupakan kelas yang berisi id, lebar, tinggi, *features*, *minimum_delay_between_moves* (mengatur jeda antar gerakan pada *bot*), *game_objects* yang berisi objek-objek permainan yang terdapat pada papan permainan yang digunakan. Pada *class* ini juga terdapat fungsi seperti *is_valid_move* untuk menentukan apakah gerakan yang dilakukan *bot valid* atau tidak.

Pada file *utils.py* yang terdapat pada folder *game/* juga terdapat beberapa fungsi sebagai berikut.

- Fungsi *clamp* untuk mengembalikan nilai *n* yang terletak di antara batas nilai terkecil dan terbesar. Jika nilai *n* lebih kecil dibandingkan batas nilai terkecil, maka akan diambil nilai dari batas terkecil tersebut, dan berlaku sebaliknya.
- Fungsi *get_direction* untuk mengembalikan nilai *delta_x* dan *delta_y* yang merupakan arah gerakan *bot* yang dilakukan jika hendak menuju suatu posisi (*destination*).
- Fungsi *position_equals* untuk menentukan apakah dua buah posisi yang dijadikan sebagai parameter merupakan posisi yang sama atau bukan.

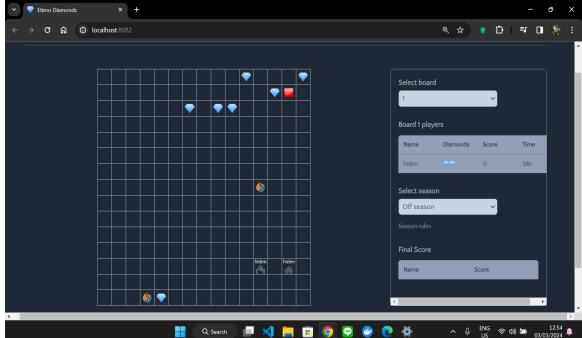
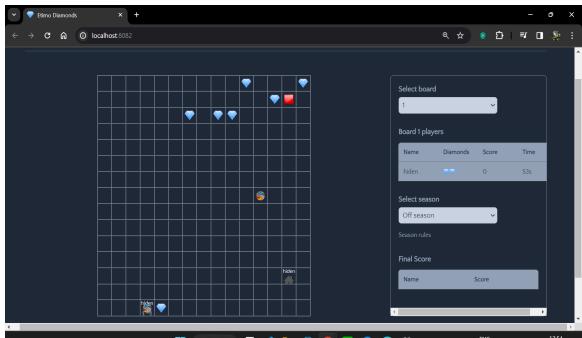
Terdapat juga file-file yang berisi *logic* yang digunakan dalam melakukan implementasi alternatif *greedy* yang digunakan sebagai otak daripada *bot* yang akan dimainkan. File ini terdapat pada folder *game/logic/*. File ini berisi fungsi *next_move()* yang merupakan fungsi yang menerapkan alternatif *greedy* untuk menentukan *goal_position*, kemudian *goal_position* akan diproses fungsi *get_direction* yang di-*import* dari file *utils.py* untuk mendapatkan arah yang akan dituju oleh *bot* untuk menuju ke *goal_position* tersebut.

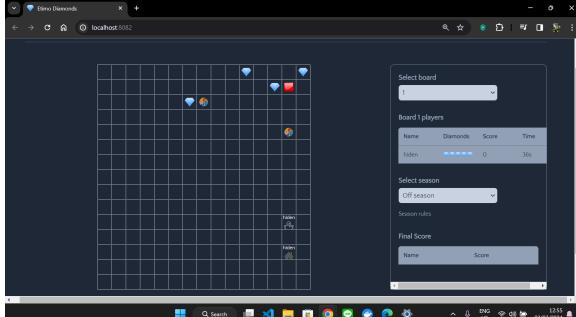
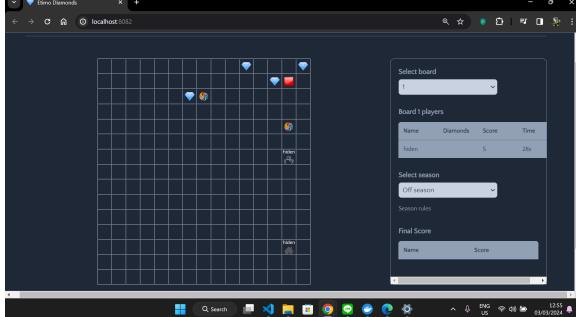
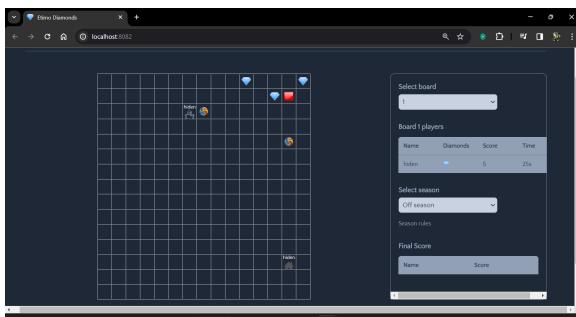
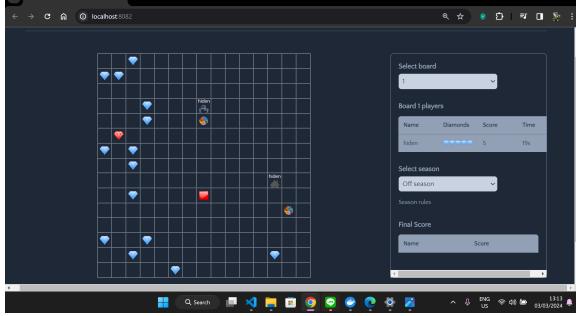
File *main.py* yang terdapat pada folder *game/* berisi algoritma utama yang digunakan untuk menghubungkan semua yang terdapat pada *bot starter pack* ini. Kita dapat menambahkan

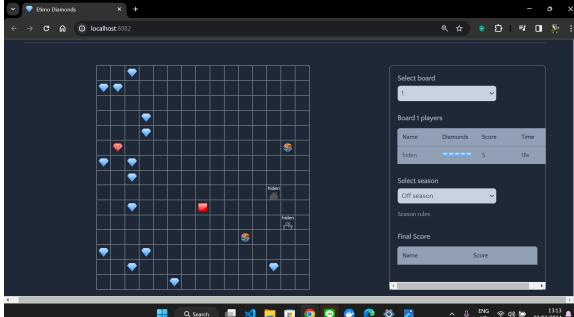
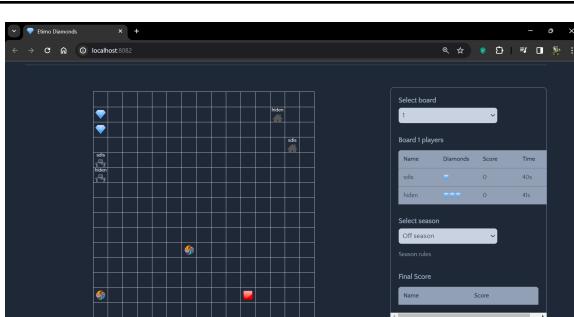
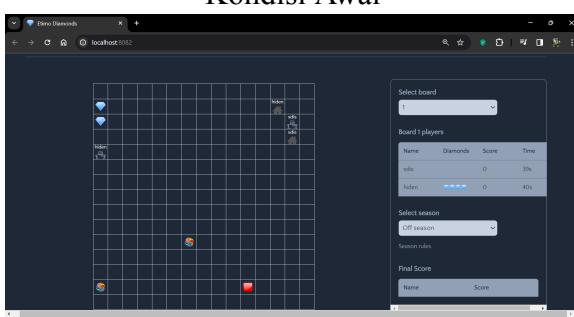
logika yang telah diimplementasikan pada folder *game/logic/* dengan melakukan *import* terhadap alternatif tersebut dan kemudian diletakkan pada *CONTROLLERS* yang terdapat pada file ini.

4.3. Analisis dan Pengujian

Pada bagian ini akan dilakukan pengujian terhadap 4 fitur utama, yakni pengambilan *diamond* berdasarkan algoritma *greedy by highest density*, pergerakan *bot* menuju *base*, penggunaan *teleporter* untuk menentukan jarak optimal menuju *diamond* ataupun *base*, serta implementasi dari fitur tambahan *greedy by collision*.

No	Gambar	Penjelasan
1.	 <p>Kondisi Awal</p>  <p>Kondisi Akhir</p>	<p>Pada kondisi awal, bot akan melakukan kalkulasi dan perbandingan untuk mengambil <i>diamond</i> dengan densitas tertinggi dari posisi bot pada saat tersebut. Dalam implementasinya, bot juga dapat memilih untuk menggunakan <i>teleporter</i> untuk memperoleh <i>diamond</i> yang ditujuinya apabila densitas <i>diamond</i> yang didapatkan dari <i>teleporter</i> lebih besar daripada tanpa menggunakan <i>teleporter</i>. Pada kondisi akhir, ditunjukkan bahwa bot telah memilih untuk menggunakan <i>teleporter</i> untuk memperoleh <i>diamond</i> dengan densitas terbesar. Hal ini menandakan keberhasilan dari fitur 1 dan 3 yakni pengambilan <i>diamond</i> berdasarkan <i>highest density</i> dan penggunaan portal untuk menentukan jarak optimal ke <i>diamond</i> yang dituju oleh bot.</p>

2.		<p>Pada gambar di samping, dapat diamati bahwa jumlah poin <i>diamond</i> yang telah diperoleh bot sejumlah 5 poin, sehingga bot memutuskan untuk kembali menuju <i>base</i> untuk menyimpannya sebagai skor. Hal ini menandakan keberhasilan dari fitur 2, pergerakan bot menuju <i>base</i>.</p>
3.	 Kondisi Awal  Kondisi Akhir	<p>Pengujian pada gambar di samping cukup serupa dengan pengujian no 1, di mana bot memilih untuk menggunakan <i>teleporter</i> untuk memperoleh <i>diamond</i> dengan densitas tertinggi dari posisinya saat itu. Hal ini menandakan keberhasilan dari fitur 1 dan 3 yakni pengambilan <i>diamond</i> berdasarkan <i>highest density</i> dan penggunaan portal untuk menentukan jarak optimal ke <i>diamond</i> yang dituju oleh bot.</p>
4.	 Kondisi Awal	<p>Pengujian pada gambar di samping cukup serupa dengan pengujian no 2, akan tetapi pada kasus di samping bot memutuskan untuk memanfaatkan <i>teleporter</i> untuk bergerak menuju <i>base</i> setelah memperoleh 5 diamond, sebagaimana dapat diamati pada kondisi awal dan akhir. Hal ini menandakan keberhasilan dari fitur 2 dan 3, yakni pergerakan bot</p>

	 <p>Kondisi Akhir</p>	<p>menuju <i>base</i> serta penggunaan <i>teleporter</i> untuk memperoleh jarak yang optimal menuju <i>base</i>.</p>
5.	 <p>Kondisi Awal</p>  <p>Kondisi Akhir</p>	<p>Pada kondisi awal, bot hiden yang menerapkan algoritma <i>greedy by highest density</i> dan bot sdis yang menerapkan algoritma <i>greedy by shortest distance</i> berjarak sebesar 1 blok. Sebagaimana implementasi dari fitur <i>collision</i>, bot hiden akan berupaya untuk melakukan <i>collision</i> apabila terdapat perbedaan jarak sebesar 1 blok, dan jumlah poin <i>diamond</i> yang dibawa kurang dari 3, serta milisekon bot hiden kurang dari milisekon bot tujuan. Pada kondisi akhir, bot hiden berhasil untuk mengimplementasikan fitur <i>collision</i> yang menyebabkan bot sdis kembali menuju base dan kehilangan poin diamond yang telah ia peroleh. Hal ini menandai keberhasilan dari fitur 4, yakni <i>greedy by collision</i> dengan teknis implementasi yang telah diuraikan pada bagian sebelumnya.</p>

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma ini, kami telah berhasil membuat bot permainan Diamonds menggunakan strategi *greedy* berdasarkan kepadatan tertinggi (*greedy by highest density*). Proses pemilihan strategi ini dibandingkan dengan strategi *greedy* lainnya dilakukan dengan mengimplementasikan beberapa strategi greedy yang berbeda dan kemudian melakukan pertandingan untuk mengetahui strategi mana yang akan dipilih berdasarkan statistik yang diperoleh.

Algoritma greedy berdasarkan densitas tertinggi yang digunakan telah terbukti efektif dibandingkan dengan alternatif greedy lainnya. Berdasarkan hasil statistik pertandingan, strategi ini berhasil mendapatkan juara 1 sebanyak dua kali, juara 2 sebanyak dua kali, dan juara 3 sebanyak satu kali dalam lima kali pertandingan. Total peserta dalam pertandingan ini adalah 4 bot, yang masing-masing menggunakan strategi greedy berdasarkan perpindahan terpendek (*greedy by shortest displacement*), konsentrasi tertinggi (*greedy by highest concentration*), densitas tertinggi (*greedy by highest density*), dan *diamond* dengan poin tertinggi *greedy by highest value* atau *diamond* merah).

Berdasarkan hasil pengamatan dan analisis terhadap pengujian yang telah diimplementasikan pada bagian sebelumnya, dapat dikatakan bahwa algoritma *greedy* yang telah diimplementasikan dapat berjalan dengan cukup optimal dengan mempertimbangkan keempat fitur utama yang meliputi pengambilan *diamond* berdasarkan algoritma *greedy by highest density*, pergerakan *bot* menuju *base*, penggunaan *teleporter* untuk menentukan jarak optimal menuju *diamond* ataupun *base*, serta implementasi dari fitur tambahan *greedy by collision*. Hasil yang diperoleh juga mempertimbangkan faktor penggunaan konfigurasi bawaan, dengan ukuran *board* 15x15, dan durasi pertandingan 1 menit.

Dengan demikian, dapat disimpulkan bahwa strategi *greedy by highest* telah berhasil untuk diimplementasikan dengan cukup optimal dengan mempertimbangkan fitur, perbandingan statistik kemenangan, serta faktor konfigurasi dari permainan.

5.2. Saran

Saran yang dapat diberikan untuk proses pengembangan *bot* untuk memenuhi tugas besar kali ini adalah sebagai berikut.

1. Lebih komunikatif antar anggota kelompok sehingga *progress* yang dibuat *bot* secara asinkron dapat tersampaikan dengan jelas kepada setiap anggota kelompok pada setiap pengembangannya,
2. Melakukan lebih banyak pertandingan atau analisis bersama dengan kelompok lain sehingga lebih mengetahui kelemahan daripada *bot* yang telah dikembangkan dibandingkan dengan hanya melakukan pertandingan antar *bot* yang dibuat sendiri,
3. Implementasi kode dapat dilakukan dengan lebih bersih sehingga tidak akan membutuhkan waktu yang cukup lama untuk melakukan proses *debugging*.

LAMPIRAN

Pranala repository

https://github.com/ChaiGans/Tubes1_TheYuds

Pranala video youtube

https://youtu.be/e_O7qF9Xmss

DAFTAR PUSTAKA

Referensi pengerjaan

- Diamonds Game Engine. Accessed March 3, 2024.
<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>.
- Munir, Rinaldi. n.d. "Algoritma Greedy (Bagian 1)." Informatika. Accessed March 3, 2024.
[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).
- Munir, Rinaldi. n.d. "Algoritma Greedy (Bagian 2)." Informatika. Accessed March 3, 2024.
[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/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf).
- Munir, Rinaldi. n.d. "Algoritma Greedy (Bagian 3)." Informatika. Accessed March 3, 2024.
[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/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf).