

LAPORAN TUGAS BESAR 01

IF2211 STRATEGI ALGORITMA

PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN DIAMONDS

Kelompok 28 : terkesTima



Disusun oleh:

Muhammad Fikri 10023519

Albert 13522081

Ivan Hendrawan Tan 13522111

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA (STEI)
INSTITUT TEKNOLOGI BANDUNG

2024

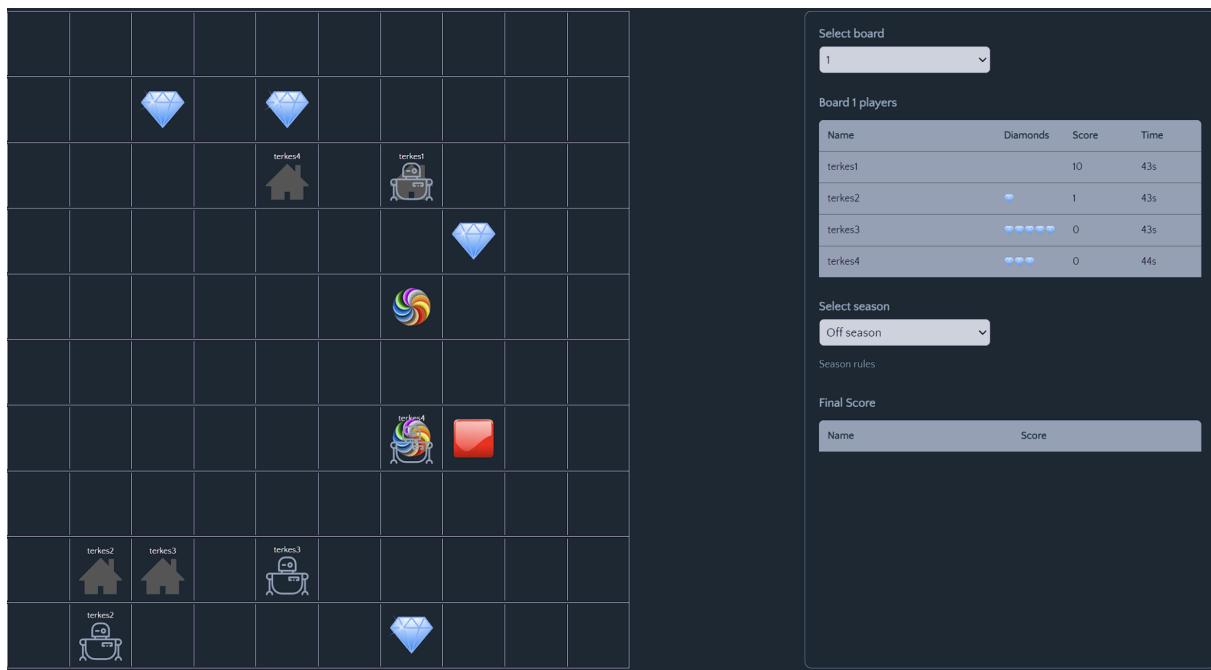
DAFTAR ISI

DAFTAR ISI.....	1
BAB I	
DESKRIPSI MASALAH.....	2
BAB II	
LANDASAN TEORI.....	6
2.1 Dasar Teori.....	6
2.2 Game Engine Diamonds.....	7
2.3 Alur Menjalankan Program.....	9
BAB III	
APLIKASI STRATEGI GREEDY.....	13
3.1. Alternatif Greedy.....	13
i. Alternatif Greedy by Shortest Diamond to Bot Distance.....	13
ii. Alternatif Greedy by Highest Density.....	15
iii. Alternatif Greedy by Highest Value.....	16
iv. Alternatif Greedy by Closest Base.....	18
3.2. Strategi Greedy yang Digunakan.....	19
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	24
4.1. Implementasi Algoritma dalam Pseudocode.....	24
4.2. Struktur Data Program.....	29
4.3. Analisis dan Pengujian.....	31
BAB V	
PENUTUP.....	34
5.1. Kesimpulan.....	34
5.2. Saran.....	34
DAFTAR REFERENSI.....	35
LAMPIRAN.....	36

BAB I

DESKRIPSI MASALAH

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



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

Untuk mengimplementasikan algoritma pada bot tersebut, user 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>

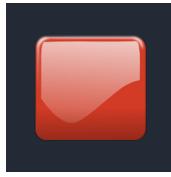
Komponen-komponen dari permainan Diamonds antara lain:

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



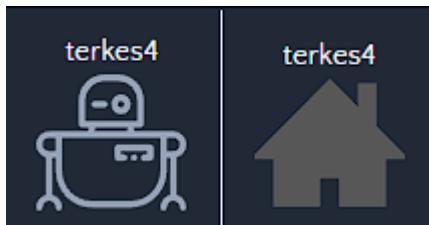
Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

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

Name	Diamonds	Score	Time
terkes1	12	9s	
terkes2	9	10s	
terkes3	5	10s	
terkes4	10	11s	

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 Dasar Teori

Algoritma *Greedy* adalah sebuah metode dalam pemrograman untuk memecahkan masalah komputasi yang mengutamakan pemilihan solusi terbaik pada setiap langkahnya tanpa mempertimbangkan konsekuensi jangka panjang. Algoritma ini selalu memilih solusi yang optimal secara lokal tanpa mempertimbangkan konsekuensi dengan harapan bahwa serangkaian pilihan yang optimal pada setiap langkah akan menghasilkan solusi global yang optimal atau mendekati optimal. Dalam algoritma *Greedy*, hanya ada dua jenis persoalan optimasi yaitu maksimasi dan minimasi.

Elemen-elemen dalam algoritma *greedy* adalah sebagai berikut.

- a. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah (misal : simpul/sisi di dalam graf, job, dll).
- 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 objektif: untuk memaksimumkan atau meminimumkan

Dalam penerapannya, algoritma *greedy* sering digunakan dalam berbagai masalah optimasi, seperti masalah jalur terpendek, masalah penjadwalan, masalah pemotongan stok, dan banyak lagi. Algoritma *Greedy* terkadang belum tentu mencapai hasil yang paling optimum karena algoritma ini tidak mengecek seluruh kemungkinan seperti metode *exhaustive search*. Meskipun algoritma *greedy* tidak selalu menjamin solusi yang optimal secara global, pendekatan ini seringkali memberikan solusi yang cukup baik dan efisien dalam hal waktu komputasi, terutama untuk masalah-masalah tertentu yang memiliki sifat *greedy-choice* property dan optimal substructure.

2.2 Game Engine Diamonds

Untuk memulai permainan dan mengimplementasikan bot ke dalam permainan Diamonds, diperlukan starter pack permainan Diamonds. Game engine digunakan untuk menampilkan peta permainan dan menyediakan arena kompetisi untuk para bot. Game engine ini telah diintegrasikan dengan docker sehingga hanya perlu mengaktifkan ruang virtual yang ada di dalam docker untuk menjalankan permainan Diamonds. Permainan ini juga memerlukan pengguna untuk menjalankan frontend dan backend yang tersedia di dalam starter pack karena permainan ini berbasis web. Starter pack yang dimaksud dapat diunduh di tautan berikut:

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

Dalam proyek GitHub untuk game Diamonds, terdapat beberapa komponen penting yang perlu dipahami agar memahami cara kerja dan logika permainan:

1. Game Engine: Ini merupakan inti dari game Diamonds, yang bertugas mengelola aspek-aspek krusial dalam permainan. Game engine bertanggung jawab atas pengaturan fisik bot, interaksi antar objek di dalam game, pengendalian logika permainan, serta interaksi antar pemain. Engine ini menjamin kelancaran permainan dan memastikan bahwa semua aturan dan mekanisme permainan berjalan sesuai dengan yang diharapkan.
2. Backend: Backend adalah bagian dari game engine yang berfokus pada pemrosesan data yang dikirim dari frontend. Backend memiliki peran penting dalam menyimpan informasi terkait jalannya game, seperti skor, posisi bot, dan lainnya. Selain itu, backend juga bertanggung jawab atas pemrosesan semua perintah yang dikirim oleh bot, termasuk pergerakan bot, pengambilan diamond, dan interaksi lainnya di dalam game.
3. Frontend: Frontend adalah komponen dari game engine yang mengatur interaksi dengan pemain. Frontend mencakup semua tampilan visual, animasi, dan hasil skor yang ditampilkan kepada pemain selama permainan berlangsung. Frontend memastikan bahwa pemain mendapatkan pengalaman bermain yang menarik dan interaktif.
4. Bot Starter Pack: Bot starter pack adalah sumber daya yang disediakan untuk memudahkan pengembangan bot dalam game Diamonds. Paket ini menyediakan template dan contoh kode yang dapat digunakan sebagai dasar dalam pembuatan bot,

sehingga pengembang dapat fokus pada pengembangan strategi dan logika bot tanpa perlu membangun dari awal.

5. Logic: Dalam bot starter pack, terdapat folder logic yang berisi semua logika bot yang dibuat. Di sini adalah tempat implementasi strategi yang diberikan kepada bot, termasuk bagaimana mengubah tujuan bot, strategi permainan, dan pengambilan keputusan selama permainan berlangsung.

Proses kerja program game Diamonds dapat dijelaskan sebagai berikut:

1. Inisialisasi Pertandingan: Ketika game engine dijalankan, pertandingan akan dimulai dengan meng-host permainan pada hostname yang telah ditentukan. Jika program dijalankan secara lokal, engine akan di-host pada localhost:8082, sehingga pemain dapat mengakses permainan melalui browser web.
2. Koneksi antara Engine dan Bot: Setelah engine berhasil dijalankan, engine akan menunggu koneksi dari bot-bot yang dijalankan oleh para pemain. Engine akan melakukan proses logging dan menyiapkan semua informasi yang diperlukan untuk pertandingan.
3. Persiapan Jumlah Bot: Jumlah bot yang akan berpartisipasi dalam satu pertandingan diatur oleh atribut BotCount yang ada dalam file "appsettings.json" di dalam folder "engine-publish". Pengaturan ini memungkinkan penyesuaian jumlah pemain dalam pertandingan sesuai dengan keinginan.
4. Memulai Pertandingan: Begitu semua bot pemain terkoneksi dan jumlahnya sesuai dengan jumlah bot yang ditetapkan, pertandingan akan dimulai. Engine akan menginisialisasi papan permainan dan menempatkan bot serta diamond di posisi awal mereka.
5. Interaksi antara Bot dan Engine: Selama pertandingan berlangsung, bot-bot akan terus berinteraksi dengan game engine. Bot akan mengirimkan perintah-perintah seperti bergerak, mengambil diamond, atau menggunakan portal. Engine akan memproses perintah tersebut dan mengirimkan informasi terbaru tentang keadaan permainan kembali ke bot.
6. Pelaksanaan Pertandingan: Pertandingan akan berlangsung hingga waktu yang ditentukan habis. Selama permainan, papan skor akan menampilkan jumlah diamond yang berhasil dikumpulkan oleh setiap bot.

Dengan struktur dan alur kerja yang terorganisir dengan baik, program Diamonds memfasilitasi interaksi antara engine dan bot pemain untuk menyelenggarakan pertandingan Diamonds yang menantang dan menyenangkan. Selain itu, sistem pencatatan hasil yang terintegrasi memungkinkan pengembang dan pemain untuk mengevaluasi kinerja bot dan mengembangkan strategi yang lebih efektif untuk pertandingan selanjutnya.

2.3 Alur Menjalankan Program

Berikut adalah langkah-langkah untuk dapat menjalankan program.

1. Pastikan Node.js, Docker, dan Yarn telah terpasang di komputer. Pemasangan Yarn dapat dilakukan dengan command berikut.

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

2. Jalankan game engine dengan mengunduh folder game engine pada tautan yang terdapat pada bagian 2.2.
 - a. Setelah berhasil mengunduh, lakukan ekstraksi file .zip tersebut dan masuk ke root dari folder hasil ekstraksi tadi, lalu buka terminal
 - b. Jalankan command berikut pada terminal untuk masuk ke root directory dari game engine

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

- c. Install dependencies dengan menggunakan yarn

```
yarn
```

- d. 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
```

- e. Setup local database dengan membuka aplikasi docker desktop, lalu masukkan command 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. Masukkan command berikut untuk melakukan proses build pada game engine

```
npm run build
```

- g. Masukkan command berikut untuk memulai game engine

```
npm run start
```

- h. Jika berhasil, maka tampilan terminal akan seperti gambar berikut

```
[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 [RouterExplorer] Mapped {/api/bots/: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/recordingss}: +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/recordingss/:seasonId, GET} route +0ms
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/recordingss/: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] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [NestApplication] Nest application successfully started +50ms
```

3. Jalankan bot starter pack dengan mengunduh folder bot starter pack pada tautan yang terdapat pada bab 1.
 - a. Setelah berhasil mengunduh, lakukan ekstraksi file .zip tersebut dan masuk ke root dari folder hasil ekstraksi tadi, lalu buka terminal
 - b. Jalankan command berikut pada terminal untuk masuk ke root directory dari bot starter pack

```
cd tubes1-IF2211-bot-starter-pack-1.0.1
```

- c. Install dependencies dengan pip

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

- d. Gunakan command berikut untuk menjalankan satu bot

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

User juga dapat menjalankan lebih dari satu bot dengan menjalankan script berikut

- Untuk Windows

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

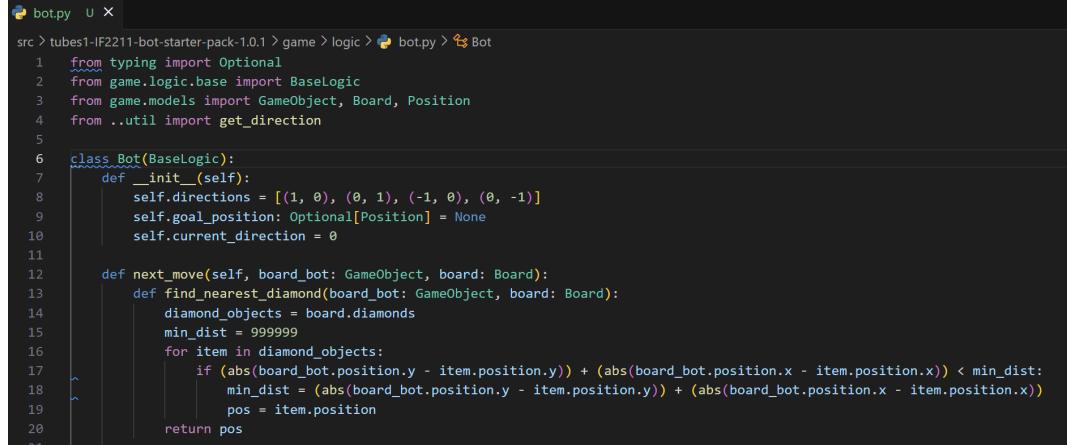
- Untuk Linux / (possibly) macOS

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

User dapat mengatur script yang ada di run-bots.bat atau run-bots.sh dari segi **logic, email, nama, dan password yang digunakan**

```
run-bots.bat
1  @echo off
2  start cmd /c "python main.py --logic Random --email=test@email.com --name=stima --password=123456 --team etimo"
3  start cmd /c "python main.py --logic Random --email=test1@email.com --name=stima1 --password=123456 --team etimo"
4  start cmd /c "python main.py --logic Random --email=test2@email.com --name=stima2 --password=123456 --team etimo"
5  start cmd /c "python main.py --logic Random --email=test3@email.com --name=stima3 --password=123456 --team etimo"
6
```

- e. Untuk merancang bot, buatlah file baru pada direktori /game/logic (misalnya *bot.py*)
- f. Buatlah sebuah kelas yang meng-inherit kelas BaseLogic, lalu implementasikan constructor beserta method next_move pada kelas tersebut

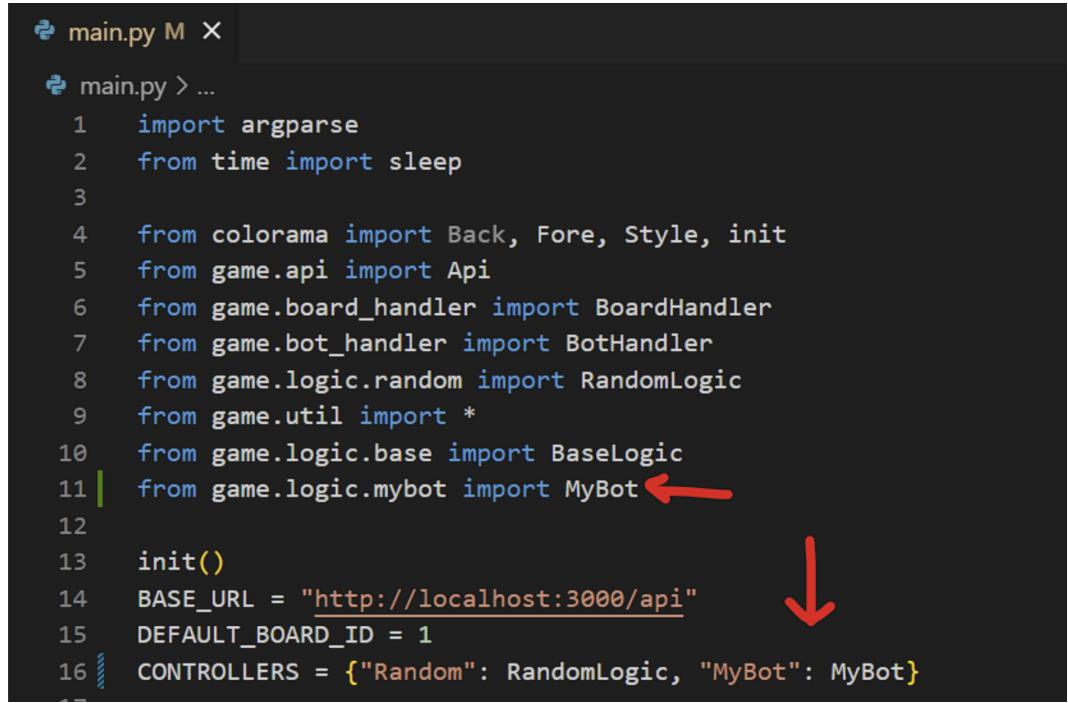


```

bot.py  U X
src > tubes1-IF2211-bot-starter-pack-1.0.1 > game > logic > bot.py > Bot
1   from typing import Optional
2   from game.logic.base import BaseLogic
3   from game.models import GameObject, Board, Position
4   from ..util import get_direction
5
6   class Bot(BaseLogic):
7       def __init__(self):
8           self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
9           self.goal_position: Optional[Position] = None
10          self.current_direction = 0
11
12      def next_move(self, board_bot: GameObject, board: Board):
13          def find_nearest_diamond(board_bot: GameObject, board: Board):
14              diamond_objects = board.diamonds
15              min_dist = 99999
16              for item in diamond_objects:
17                  if (abs(board_bot.position.y - item.position.y)) + (abs(board_bot.position.x - item.position.x)) < min_dist:
18                      min_dist = (abs(board_bot.position.y - item.position.y)) + (abs(board_bot.position.x - item.position.x))
19                      pos = item.position
20
21          return pos

```

- g. Import kelas yang telah dibuat tadi ke dalam *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}

```

- h. Jalankan program seperti pada step d pada bagian 3 dengan menyesuaikan argumen logic pada command/script dengan nama yang terdaftar pada *CONTROLLERS*. User juga masih bisa menjalankan satu bot saja atau beberapa bot dengan run-bots.bat atau run-bots.sh

BAB III

APLIKASI STRATEGI GREEDY

3.1. Alternatif Greedy

i. Alternatif Greedy by Shortest Diamond to Bot Distance

Greedy by Shortest Diamond to Bot Distance adalah sebuah alternatif strategi dimana strategi ini lebih mengedepankan jarak paling dekat diamond terhadap bot tanpa mempedulikan jenis diamondnya (warna biru atau merah). Bot hanya akan mengutamakan diamond yang berada paling dekat dengan posisi bot.

- Pemetaan Elemen *Greedy*
 - a. Himpunan kandidat: Himpunan diamond yang ada pada board, terdiri dari diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin
 - b. Himpunan solusi: Himpunan diamond yang terpilih
 - c. Fungsi solusi: Menjumlahkan keseluruhan diamond yang telah dikumpulkan oleh bot
 - d. Fungsi seleksi: Memilih diamond yang terdekat dari bot tanpa melihat besar poin yang akan didapat
 - e. Fungsi kelayakan: Memeriksa jumlah diamond yang dibawa tidak melebihi 4 poin setiap kali akan bergerak. Jika telah mencapai 4 poin, maka bot akan langsung bergerak kembali menuju base
 - f. Fungsi objektif: Jumlah diamond yang didapat maksimum
- Analisis Efisiensi Solusi

Pada alternatif ini, dilakukan inisialisasi terhadap suatu *array* kosong yang nantinya menampung keseluruhan object *diamond* yang tersedia pada *board*. Setelah mendapat semua object diamond yang disimpan di suatu array dilakukan iterasi melalui semua objek diamond yang tersedia untuk mencari diamond terdekat dengan posisi bot saat ini. Proses iterasi ini memiliki kompleksitas $O(n)$, dimana n adalah jumlah diamond. Untuk setiap diamond, dilakukan perhitungan jarak

antara bot dan diamond tersebut, yang memiliki kompleksitas $O(1)$. Setelah semua jarak dihitung, dilakukan pencarian diamond dengan jarak minimum, yang juga memiliki kompleksitas $O(n)$. Dengan demikian, total kompleksitas dari fungsi mencari jarak terdekat adalah $O(n)$, yang berarti waktu yang dibutuhkan untuk menjalankan fungsi ini bertambah seiring dengan bertambahnya jumlah diamond.

Dalam strategi ini, terdapat beberapa operasi utama, termasuk pencarian diamond terdekat yang telah dijelaskan sebelumnya dengan kompleksitas $O(n)$. Selain itu, terdapat juga pencarian teleporter terdekat, yang mirip dengan pencarian diamond terdekat, dan memiliki kompleksitas $O(t)$, dimana t adalah jumlah teleporter. Operasi lainnya, seperti perhitungan jarak dan pemilihan tujuan, memiliki kompleksitas konstan $O(1)$. Jika kita asumsikan bahwa jumlah teleporter juga sebanding dengan jumlah diamond n , maka kompleksitas keseluruhan dari program adalah $O(n)$. Ini berarti waktu yang dibutuhkan untuk menjalankan fungsi ini juga bertambah seiring dengan bertambahnya jumlah objek pada papan permainan.

- Analisis Efektivitas Solusi

Strategi ini mengutamakan jarak terdekat (*shortest distance*) *bot* menuju *diamond*. Strategi ini menjadi efektif apabila:

- a. Bot *spawn* jauh dari bot lain, mengurangi risiko bertabrakan dengan bot lain.
- b. Bot *spawn* di area dengan konsentrasi diamond yang tinggi, memungkinkan bot untuk mengumpulkan diamond dengan cepat dan kembali ke base dengan efisien.
- c. Adanya portal yang mempersingkat jarak tempuh, mempercepat perjalanan bot menuju diamond atau kembali ke base.

Strategi ini menjadi tidak efektif apabila:

- a. Jarak yang sangat dekat antara bot dan bot lawan, yang meningkatkan kemungkinan terjadi tabrakan.

-
- b. Kehadiran teleporter yang menghalangi jalur bot, sehingga membuat bot mencari ulang kemungkinan diamond lain yang memiliki jarak paling dekat.

ii. Alternatif Greedy by Highest Density

Greedy by highest density adalah sebuah strategi *greedy* yang mengutamakan diamond yang memiliki value poin per jarak terbesar. Bot ini akan menghitung terlebih dahulu besarnya poin per jarak dari setiap diamond yang ada di papan permainan dan memilih diamond yang memiliki value poin per jarak yang terbesar,

- Pemetaan Elemen *Greedy*
 - a. Himpunan kandidat: Himpunan diamond pada board
 - b. Himpunan solusi: Himpunan diamond yang poinnya telah dibagi sebesar jaraknya dari bot
 - c. Fungsi solusi: Menghitung total poin diamond yang telah dikumpulkan
 - d. Fungsi seleksi: Memilih diamond yang memiliki nilai poin per jarak terbesar
 - e. Fungsi kelayakan: Memeriksa jumlah diamond yang dibawa oleh bot tidak melebihi 4 poin setiap kali bot bergerak dan akan pulang ke base jika telah mencapai 4 poin
 - f. Fungsi objektif: Jumlah poin dari diamond yang didapat besarnya maksimum

- Analisis Efisiensi Solusi

Pada alternatif ini, dilakukan inisialisasi terhadap suatu array kosong yang nantinya menampung keseluruhan object diamond yang tersedia pada papan permainan. Setelah mendapat semua object diamond, iterasi seluruh objek diamond yang ada pada array. Proses iterasi ini memiliki kompleksitas $O(n)$, dimana n adalah jumlah diamond. Untuk setiap diamond, dilakukan perhitungan jarak antara diamond dengan bot beserta menghitung kepadatan dari diamond, yang masing-masing memiliki kompleksitas $O(1)$. Setelah itu, dilakukan

kembali iterasi untuk mencari diamond yang memiliki densitas terbesar dengan kompleksitas $O(n)$.

Dalam strategi ini, terdapat beberapa operasi lainnya, seperti pencarian teleporter terdekat, yang mirip dengan pencarian diamond dengan densitas terbesar, dan memiliki kompleksitas $O(t)$, dimana t adalah jumlah teleporter. Operasi lainnya, seperti perhitungan jarak dan pemilihan tujuan, memiliki kompleksitas konstan $O(1)$. Jika kita asumsikan bahwa jumlah teleporter juga sebanding dengan jumlah diamond n , maka kompleksitas keseluruhan dari program adalah $O(n)$.

- Analisis Efektivitas Solusi

Strategi ini mengutamakan area yang memiliki diamond terbanyak dalam radius tertentu. Strategi ini akan efektif bila:

- a. Terdapat diamond merah yang sedikit lebih jauh dari bot namun masih memiliki value poin per jarak yang lebih besar dibandingkan diamond biru yang lebih dekat terhadap bot.
- b. Adanya portal yang mempersingkat jarak tempuh sehingga mempercepat perjalanan bot menuju diamond dan/atau kembali ke base.

Strategi ini dapat menjadi tidak efektif bila:

- a. Ada diamond merah dan biru yang sama-sama memiliki value poin per jarak yang sama tetapi jarak diamond merah yang lebih jauh.
- b. terdapat teleporter yang menghalangi jalur bot, sehingga membuat jarak diamond sedikit lebih jauh.

iii. Alternatif Greedy by Highest Value

Greedy by highest value adalah salah satu strategi *greedy* yang mengutamakan diamond merah yang bernilai 2 poin dibandingkan diamond biru yang hanya 1 poin. Bot akan bergerak menuju diamond merah terdekat selama masih ada di papan permainan. Jika tidak ada diamond merah, bot akan mencari diamond biru terdekat dalam papan permainan.

- Pemetaan Elemen *Greedy*

- a. Himpunan kandidat: Himpunan diamond merah dan diamond biru dalam papan permainan
- b. Himpunan solusi: Himpunan diamond yang terpilih
- c. Fungsi solusi: Menjumlahkan seluruh poin dari diamond yang telah dikumpulkan
- d. Fungsi seleksi: Memilih diamond merah yang terdekat
- e. Fungsi kelayakan: Memeriksa jumlah diamond yang dibawa oleh bot tidak melebihi 4 poin setiap kali bot bergerak dan akan pulang ke base jika telah mencapai 4 poin
- f. Fungsi objektif: Jumlah poin dari diamond yang didapat besarnya maksimum

- Analisis Efisiensi Solusi

Untuk alternatif ini, dilakukan inisialisasi terhadap 2 buah array yang akan menampung diamond merah dan diamond biru yang ada pada papan permainan. Untuk menghasilkan array diamond merah dan array diamond biru, diperlukan iterasi sebanyak $2n$ kali sehingga iterasi ini memiliki kompleksitas $O(n)$. Untuk setiap diamond merah dan biru, dilakukan pencarian jarak diamond dan bot dengan kompleksitas $O(1)$. Setelah seluruh jarak antara diamond dan bot diketahui, dilakukan pencarian jarak terdekat dengan kompleksitas $O(n)$.

Dalam strategi ini, terdapat beberapa operasi lainnya seperti pencarian teleporter terdekat, yang mirip dengan pencarian diamond terdekat, dan memiliki kompleksitas $O(t)$, dimana t adalah jumlah teleporter. Ada juga perhitungan jarak dan pemilihan tujuan, memiliki kompleksitas konstan $O(1)$. Jika kita asumsikan bahwa jumlah teleporter juga sebanding dengan jumlah diamond n , maka kompleksitas keseluruhan dari program adalah $O(n)$.

- Analisis Efektivitas Solusi

Strategi ini mengutamakan diamond merah terdekat, lalu diamond biru terdekat. Strategi ini menjadi efektif apabila:

- a. Bot *spawn* jauh dari bot lain, mengurangi risiko bertabrakan dengan bot lain.

- b. Bot *spawn* di area dekat dengan diamond merah
- c. Adanya portal yang mempersingkat jarak tempuh, mempercepat perjalanan bot menuju diamond merah atau kembali ke base.

Strategi ini menjadi tidak efektif apabila:

- a. Tidak adanya diamond merah di papan permainan
- b. Jarak yang sangat dekat antara bot dan bot lawan, yang meningkatkan kemungkinan terjadi tabrakan.
- c. Kehadiran portal yang menghalangi jalur bot, sehingga membuat bot mencari ulang kemungkinan diamond lain yang memiliki jarak paling dekat.

iv. Alternatif Greedy by Closest Base

Greedy by Closest Base adalah salah satu strategi *greedy* yang mengedepankan jarak diamond terdekat terhadap base. Bot akan memilih diamond berdasarkan perhitungan jaraknya terhadap base dari bot tersebut.

- Pemetaan Elemen *Greedy*
 - a. Himpunan kandidat: Himpunan diamond pada board
 - b. Himpunan solusi: Himpunan diamond yang terpilih
 - c. Fungsi solusi: Menghitung seluruh diamond yang telah dikumpulkan
 - d. Fungsi seleksi: Memilih diamond yang terdekat dari base
 - e. Fungsi kelayakan: Memeriksa jumlah diamond yang dibawa oleh bot tidak melebihi 4 poin setiap kali bot bergerak dan akan pulang ke base jika telah mencapai 4 poin
 - f. Fungsi objektif: Jumlah poin dari diamond yang didapat besarnya maksimum

- Analisis Efisiensi Solusi

Pada alternatif ini, dilakukan inisialisasi terhadap suatu array kosong yang nantinya menampung keseluruhan object diamond yang tersedia pada papan permainan. Setelah mendapat semua object diamond, iterasi seluruh objek diamond yang ada pada array. Proses

iterasi ini memiliki kompleksitas $O(n)$, dimana n adalah jumlah diamond. Setelah itu, dilakukan pengurutan dengan fungsi bawaan python sorted() yang memiliki kompleksitas $O(n \log n)$ untuk kasus terburuk dan rata-rata.

Dalam strategi ini, terdapat beberapa operasi lainnya, seperti pencarian teleporter terdekat, yang mirip dengan pencarian diamond dengan densitas terbesar, dan memiliki kompleksitas $O(t)$, dimana t adalah jumlah teleporter. Operasi lainnya, seperti perhitungan jarak dan pemilihan tujuan, memiliki kompleksitas konstan $O(1)$. Jika kita asumsikan bahwa jumlah teleporter juga sebanding dengan jumlah diamond n , maka kompleksitas keseluruhan dari program adalah $O(n)$.

- Analisis Efektivitas Solusi

Strategi ini sangat mengutamakan diamond yang jaraknya paling dekat dari base milik bot. Strategi ini akan sangat efektif bila:

- a. Banyak diamond yang muncul di sekitar base.
- b. Base milik bot berada di tengah-tengah map

Strategi ini menjadi tidak efektif apabila:

- a. Banyak diamond yang jaraknya jauh dari base.
- b. Diamond yang jaraknya lebih dekat dengan bot dapat terabaikan
- c. Jarak yang dekat antara bot ini dan bot lawan, yang meningkatkan kemungkinan terjadinya tabrakan.

3.2. Strategi Greedy yang Digunakan

Berdasarkan evaluasi yang dilakukan terhadap berbagai alternatif strategi *greedy* yang telah dijelaskan pada bagian 3.1, kelompok kami memutuskan untuk mengadopsi strategi *Greedy by Highest Density* sebagai strategi utama dalam pengembangan bot kami. Keputusan ini didasarkan pada pertimbangan bahwa durasi pertandingan Diamonds adalah 60 detik, dengan asumsi satu gerakan per detik. Dalam kondisi seperti ini, efektivitas dan efisiensi dalam pengambilan diamond menjadi sangat krusial. Strategi *Greedy by Highest Density* memungkinkan bot untuk fokus pada diamond yang memiliki nilai poin per jarak terbesar. Strategi ini juga memiliki keunggulan

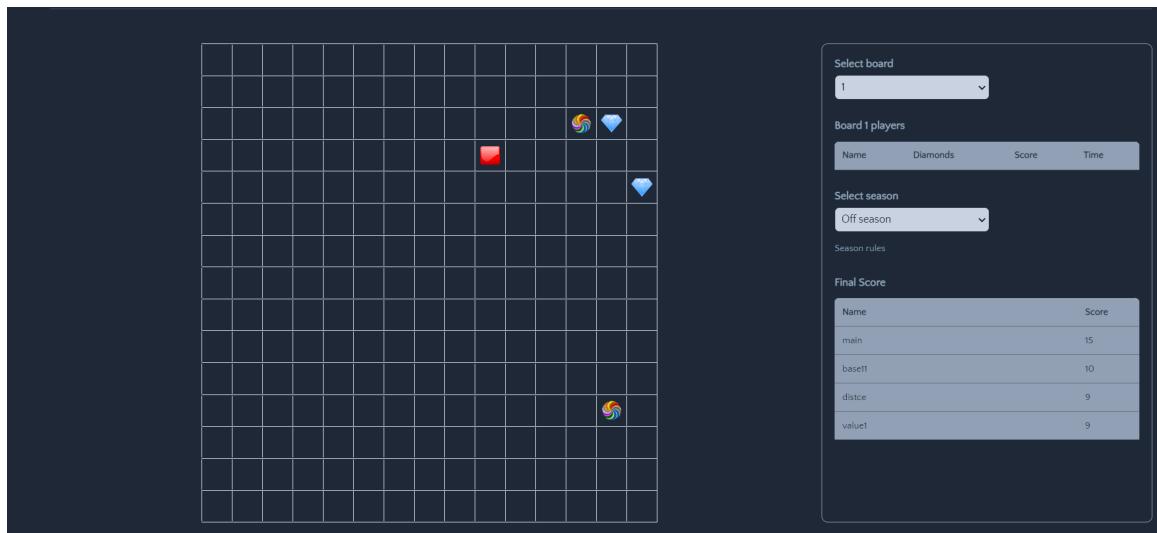
dalam mengoptimalkan total poin yang diperoleh bot. Dengan memprioritaskan diamond yang memiliki nilai poin per jarak terbesar, bot dapat mengumpulkan poin dengan lebih efisien. Hal ini memberikan keuntungan kompetitif yang signifikan dalam pertandingan, memungkinkan bot untuk meraih kemenangan dengan skor yang maksimal dalam waktu yang singkat. Mengingat alasan-alasan tersebut, strategi *Greedy by Highest Density* terbukti menjadi pilihan yang efektif dan efisien untuk bot kami dalam permainan Diamonds yang berlangsung selama 60 detik. Strategi ini memaksimalkan peluang bot kami untuk memenangkan pertandingan dengan memanfaatkan setiap kesempatan yang ada untuk mengumpulkan poin secara strategis.

Strategi *Greedy by Highest Value* tidak dipilih sebagai strategi utama karena seringkali bot harus melakukan perjalanan yang cukup jauh untuk mendapatkan diamond merah dengan nilai poin dua. Hal ini kerap kali mengakibatkan bot kehabisan waktu dalam perjalanan kembali ke base untuk menyimpan diamond, sehingga tidak ada kesempatan lagi untuk mencari diamond lain.

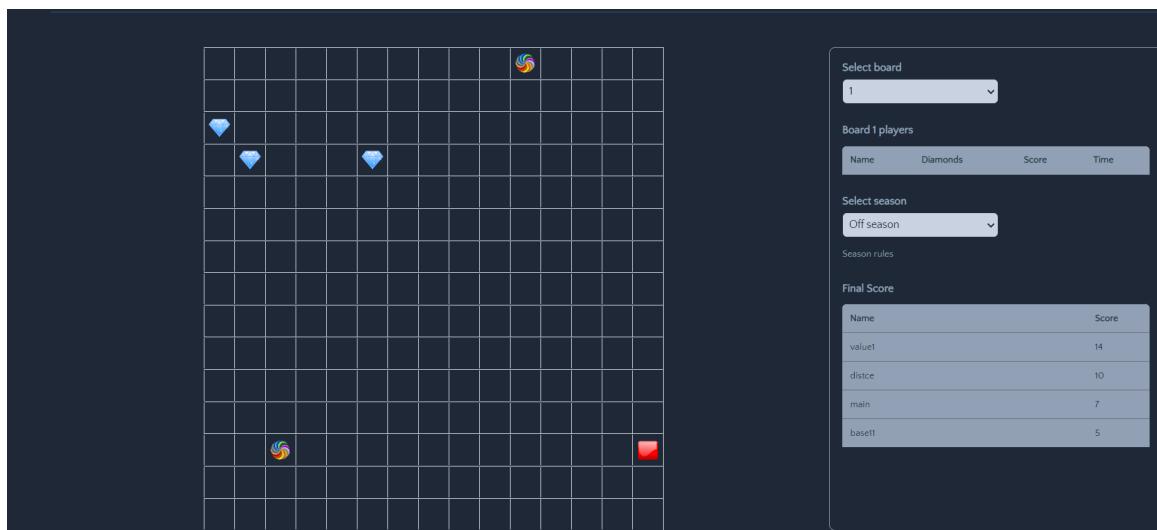
Strategi *Greedy by Shortest Diamond to Bot Distance* juga tidak diadopsi karena strategi ini tidak mempertimbangkan nilai poin dalam pengambilan diamond. Walaupun solusi ini memiliki kemiripan dengan strategi *Greedy by Density*, namun karena hanya fokus pada jarak tanpa memperhitungkan nilai diamond, bot terkadang cenderung memilih diamond biru yang hanya bernilai satu poin meskipun ada diamond merah dengan nilai yang sama dalam jarak yang sama. Hal ini mengurangi efisiensi dalam mengumpulkan poin. Sedangkan strategi *Greedy by Closest Base* tidak dipilih karena fokus utamanya adalah pada diamond yang paling dekat dengan *base*, yang terkadang menyebabkan bot melewatkkan diamond yang lebih dekat dengannya. Dengan strategi ini, bot cenderung berputar-putar di sekitar base untuk mengambil diamond yang terdekat dengan base, tanpa mempertimbangkan efisiensi dalam mengumpulkan poin dari diamond yang lebih dekat dengannya.

Berikut merupakan hasil pertandingan antar bot:

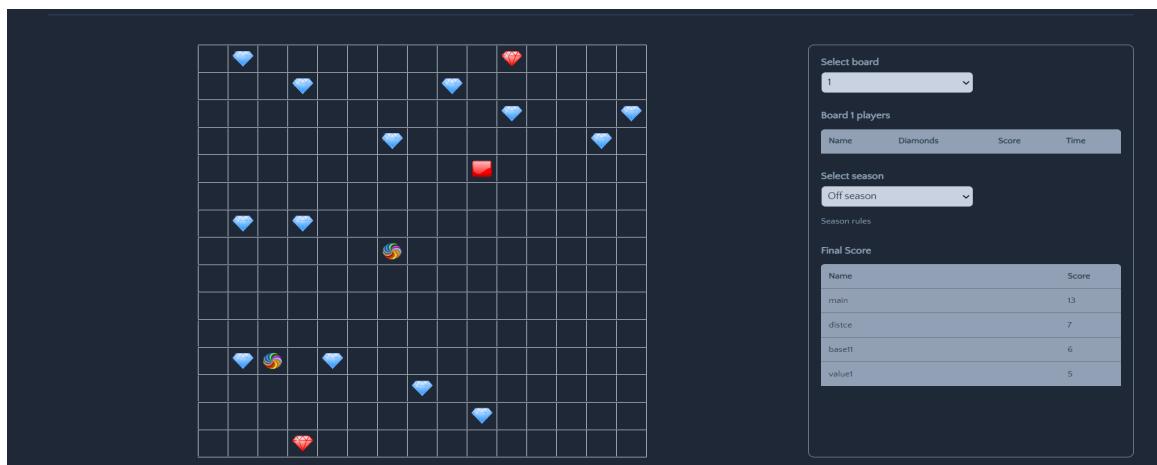
1. Pertandingan 1



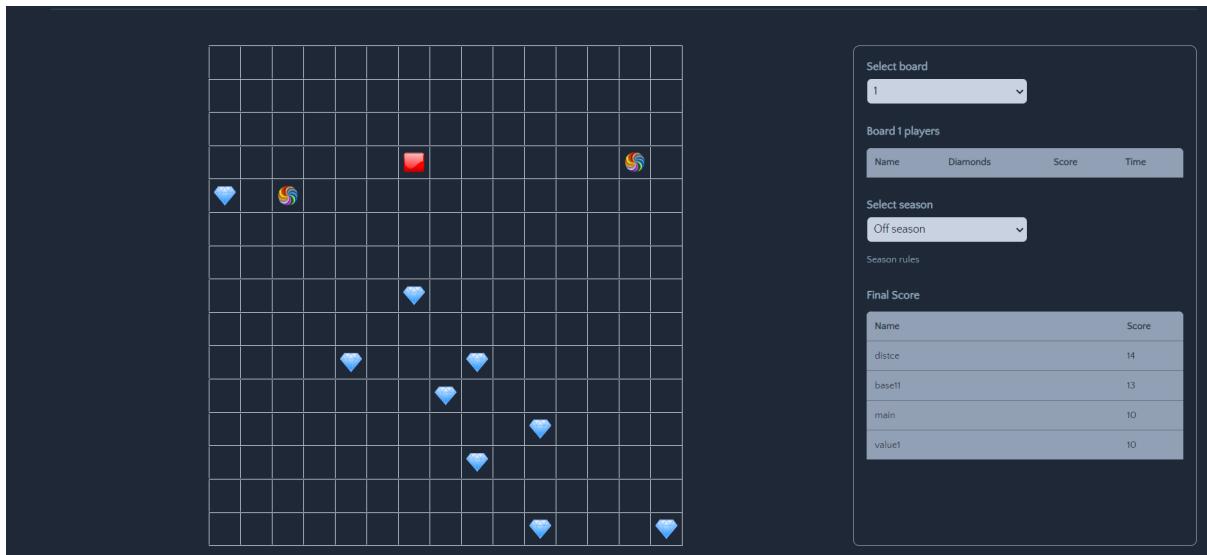
2. Pertandingan 2



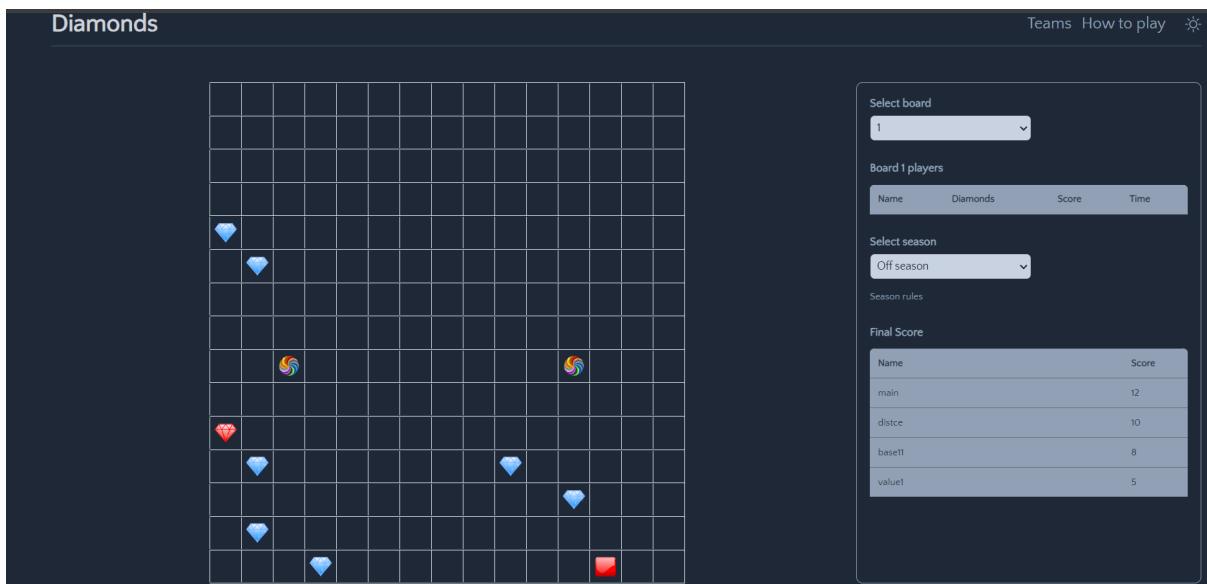
3. Pertandingan 3



4. Pertandingan 4



5. Pertandingan 5



Dari hasil lima pertandingan yang telah dilaksanakan, terlihat adanya variasi pemenang pada setiap game dengan beberapa strategi yang berbeda. Bot dengan strategi *Greedy by Highest Density* (bot main) menunjukkan performa yang paling impresif, dengan meraih posisi juara pertama sebanyak tiga kali dan juara ketiga sebanyak dua kali. Hal ini menunjukkan bahwa bot main merupakan bot yang paling efisien dan efektif, serta menunjukkan konsistensi yang tinggi dalam berada di tiga besar pada setiap pertandingan, meskipun kondisi papan permainan berbeda-beda.

Strategi *Greedy by Shortest Distance* juga menunjukkan hasil yang cukup baik dengan seringnya berada di posisi juara kedua dan ketiga. Namun, strategi ini belum kami pilih sebagai strategi utama karena belum optimal dalam hal pengambilan diamond, terutama ketika terdapat diamond merah dan biru yang berada pada jarak yang sama. Sementara itu, strategi *Greedy by Base* menawarkan pendekatan yang menarik dengan mengutamakan diamond yang paling dekat dengan base. Strategi ini dapat sangat menguntungkan jika terdapat banyak diamond dengan nilai tinggi yang terletak dekat dengan base, sehingga meminimalkan waktu yang dibutuhkan untuk mengumpulkan diamond. Namun, konsistensi strategi ini kurang terjamin karena sangat bergantung pada keberadaan diamond di sekitar base. Jika tidak terdapat diamond di dekat base, bot akan cenderung menghabiskan waktu untuk mengejar bot lain. Dengan demikian, berdasarkan analisis dan hasil pertandingan, strategi *Greedy by Highest Density* terpilih sebagai strategi utama karena menunjukkan kinerja yang konsisten dan efektif dalam berbagai kondisi permainan. Kode logic dari bot main yang kita gunakan berada di dalam folder logic dengan nama *mybot.py*.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma dalam Pseudocode

Fungsi calculate_distance

```
{ Fungsi ini merupakan fungsi untuk menghasilkan integer , dimana fungsisi akan  
menerima 2 buah parameter bertipe position.}
```

```
function calculate_distance(a: Position, b: Position) → integer  
→ abs(a.x - b.x) + abs(a.y - b.y)
```

Fungsi direction_correcter

```
{ Fungsi ini merupakan fungsi yang menghasilkan 2 integer , dimana fungsi akan  
menerima 2 buah parameter bertipe integer.}
```

```
function direction_correcter(deltax: integer, deltay: integer) -> integer,  
integer  
if (deltax = 0 and deltay = 0) then  
    delta_x, delta_y ← random.choice([(1, 0), (0, 1), (-1, 0), (0, -1)])  
    → delta_x, delta_y  
→ deltax, deltay
```

Fungsi find_nearest_diamond

```
{ Fungsi ini merupakan fungsi yang menghasilkan sebuah position, dimana fungsi  
akan menerima 2 buah parameter bertipe position dan list yang berisi diamond.}
```

```
function find_nearest_diamond(current: Position, diamonds: List[GameObject]) →  
Optional[Position]  
if not(diamonds) then  
    → None  
nearest_diamond ← min(diamonds, key=lambda diamond:  
calculate_distance(current, diamonds.position))  
→ nearest_diamond.postion
```

Fungsi `find_nearest_teleporter_pair`

```
{ Fungsi ini merupakan fungsi yang menghasilkan sebuah tuple yang berisi 2 position, dimana fungsi akan menerima 2 buah parameter bertipe position dan list yang berisi teleporter.}
```

```
function      find_nearest_teleporter_pair(current:      Position,      teleporters:
List[GameObject] → Optional[Tuple[Position, Position]]
    nearest_teleporter_pair ← None
    min_distance ← float("inf")

    teleporter_pairs ← {}
    for teleporter in teleporters do
        pair_id ← teleporter.properties.pair_id
        if pair_id not in teleporter_pairs then
            teleporter_pairs[pair_id] = []
        teleporter_pairs[pair_id].append(teleporter)

    for pair_id, pair in teleporter_pairs.items() do
        if len(pair) = 2 then
            teleporter, paired_teleporter ← pair
            distance ← calculate_distance(current, teleporter.position)
            if distance < min_distance then
                min_distance ← distance
                nearest_teleporter_pair      =      (teleporter.position,
                                                paired_teleporter.position)
        → nearest_teleporter_pair
```

Fungsi `get_direction_pribadi`

```
{ Fungsi ini merupakan fungsi yang untuk mengoreksi direction yang dihasilkan agar tidak error, dimana fungsi akan menerima 4 buah parameter bertipe integer dan sebuah list yang berisi position dari teleporter }
```

```
def get_direction_pribadi(current_x, current_y, dest_x, dest_y,
                           avoid_teleporters=[]):
    delta_x ← clamp(dest_x - current_x, -1, 1)
    delta_y ← clamp(dest_y - current_y, -1, 1)

    # Menghindar dari teleporter
    if (current_x + delta_x, current_y + delta_y) in avoid_teleporters and (dest_x,
                           dest_y) not in avoid_teleporters:
        alternative_moves ← [(1, 0), (0, 1), (-1, 0), (0, -1)]
        alternative_moves.remove((delta_x, delta_y))
```

```

        for move in alternative_moves:
            if (current_x + move[0], current_y + move[1]) not in
                avoid_teleporters:
                return move
            return (delta_x, delta_y)

    else:
        if delta_x != 0:
            delta_y ← 0
        return (delta_x, delta_y)

```

Fungsi is_on_path_or_closed

{ Fungsi ini merupakan fungsi yang menghasilkan sebuah integer, dimana fungsi akan menerima 3 buah parameter bertipe position, position, dan literal. }

```

function is_on_path_or_closed(diamond_position: Position, bot_position: position,
threshold: literal → int
    close_to_path ← calculate_distance(diamond_position, bot_position) ≤
    threshold
    → close_to_path

```

Fungsi next_move

{ Fungsi ini merupakan fungsi yang menghasilkan 2 integer, dimana fungsi akan menerima 3 buah parameter bertipe self, GameObject, dan Board. }

```

function next_move (self, board_bot: GameObject, board: Board) → int, int
    props ← board_bot.properties
    base ← props.base
    radius ← board.height / 2
    time_left ← props.milliseconds_left / 1000
    bot_position ← board_bot.position

    diamond_game_objects ← []
    teleport_game_objects ← []
    diamond_button_game_objects ← []
    for obj in board.game_objects do
        if obj.type = "DiamondGameObject" then
            diamond_game_objects.append(obj)
        else

```

```

    if obj.type = "TeleportGameObject" then
        teleport_game_objects.append(obj)
    else
        if obj.type = "DiamondButtonGameObject" then
            diamond_button_game_objects.append(obj)

    for diamond in diamond_game_objects do
        distance ← max(1, calculate_distance(bot_position, diamond.position))
        diamond.density ← diamond.properties.points / distance

        highest_density_diamond ← max(diamond_game_objects, key=lambda d: d.density, default ← None)

        time_to_reach_base ← calculate_distance(board_bot.position, base)
        nearest_diamond ← find_nearest_diamond(bot_position, diamond_game_objects)
        nearest_teleporter_pair ← find_nearest_teleporter_pair(bot_position, teleport_game_objects)
        if nearest_teleporter_pair then
            teleporter1, teleporter2 ← nearest_teleporter_pair
            if calculate_distance(bot_position, teleporter1) ≤ calculate_distance(bot_position, teleporter2) then
                nearest_teleporter ← teleporter1
                paired_teleporter ← teleporter2
            else
                nearest_teleporter ← teleporter2
                paired_teleporter ← teleporter1
        if props.diamonds ≥ props.inventory_size - 1 then
            distance_to_base ← calculate_distance(bot_position, base)
            distance_to_base_via_teleport ← calculate_distance(paired_teleporter, base) + calculate_distance(bot_position, nearest_teleporter)
            if distance_to_base_via_teleport < distance_to_base then
                self.goal_position ← nearest_teleporter
            else
                path_diamonds = [diamond for diamond in diamond_game_objects if is_on_path_or_close(diamond.position, bot_position, threshold=3) and diamond.properties.points + props.diamonds ≤ props.inventory_size]
                if time_left < time_to_reach_base then
                    self.goal_position ← base
                else
                    if path_diamond then
                        nearest_path_diamond ← find_nearest_diamond(bot_position, path_diamonds)
                        self.goal_position ← nearest_path_diamond
                    else

```

```

        self.goal_position ← base
    else
        if (props.diamonds ≥ 2 and time_left < time_to_reach_base + 3) then
            distance_to_base ← calculate_distance(bot_position, base)
            distance_to_base_via_teleport ← calculate_distance(paired_teleporter,
            base) + calculate_distance(bot_position, nearest_teleporter)
            if distance_to_base_via_teleport < distance_to_base then
                self.goal_position ← nearest_teleporter
            else
                self.goal_position ← base
        else
            if not any(diamond for diamond in diamond_game_objects if
            calculate_distance(diamond.position, base) ≤ radius) then
                nearest_diamond_button = find_nearest_diamond(bot_position,
                diamond_button_game_objects)
                if calculate_distance(bot_position, nearest_diamond) ≤
                calculate_distance(bot_position, nearest_diamond_button) then
                    self.goal_position ← nearest_diamond
                else
                    distance_to_diamond_button ← calculate_distance(bot_position,
                    nearest_diamond_button)
                    if nearest_teleporter and distance_to_diamond_button >
                    calculate_distance(paired_teleporter, nearest_diamond_button) +
                    calculate_distance(bot_position, nearest_teleporter) then
                        self.goal_position ← nearest_teleporter
                    else
                        self.goal_position = nearest_diamond_button
                else
                    distance_to_diamond ← calculate_distance(bot_position,
                    highest_density_diamond.position)
                    if nearest_teleporter and distance_to_diamond >
                    calculate_distance(paired_teleporter,
                    highest_density_diamond.position) + calculate_distance(bot_position,
                    nearest_teleporter) then
                        self.goal_position ← nearest_teleporter
                    else
                        self.goal_position ← highest_density_diamond.position
            if self.goal_position then
                if bot_position = nearest_teleporter and self.goal_position =
                nearest_teleporter then
                    self.goal_position ← base
                    delta_x, delta_y ← get_direction_pribadi(bot_position.x,
                    bot_position.y, self.goal_position.x, self.goal_position.y)
            if delta_x = 0 and delta_y = 0 then

```

```
delta_x, delta_y ← direction_correcter(delta_x, delta_y)
→ delta_x, delta_y
```

4.2. Struktur Data Program

Bahasa pemrograman yang digunakan untuk implementasi bot adalah bahasa pemrograman Python. Struktur data program didefinisikan di dalam class pada file *models.py*. Class yang digunakan dalam implementasi dan pengembangan bot terdapat di dalam folder game.

Class yang terdapat di dalam *models.py* adalah sebagai berikut.

- Class bot adalah sebuah class yang berisi informasi mengenai bot terkait nama, email, dan juga id bot tersebut
- Class position merepresentasikan posisi dari objek yang terdiri dari koordinat x dan y
- Class base merupakan kelas turunan dari position yang berisi informasi mengenai koordinat x dan y untuk base dari sebuah bot
- Class properties menjelaskan berbagai properti dari objek yang ada di dalam permainan sebagai berikut
 - points merupakan properti milik diamond yang berisi informasi mengenai jumlah poin dari setiap jenis diamond, diamond biru memiliki poin 1 dan diamond merah memiliki poin 2,
 - pair_id merupakan properti milik teleporter yang berisi id untuk teleporter agar dapat dilakukan pairing,
 - diamonds merupakan informasi dari jumlah diamond yang dibawa oleh bot,
 - score merupakan informasi dari jumlah poin dari diamond yang berhasil dibawa bot ke base,
 - name adalah informasi mengenai nama dari objek tersebut,
 - inventory_size merupakan properti milik bot yang berisi jumlah poin maksimal yang dapat dibawa oleh bot,
 - can_tackle berisi informasi apakah sebuah objek dapat ditabrak atau tidak,

- miliseconds_left berisi informasi mengenai sisa waktu yang dimiliki oleh bot di dalam permainan,
 - time_joined berisi informasi mengenai waktu masuk dari sebuah objek ke dalam permainan,
 - base berisi informasi mengenai koordinat dari base sebuah bot.
- Class GameObject merupakan sebuah kelas yang berisi informasi mengenai objek seperti id, position, type, dan properties. Id bertipe integer, position bertipe class postion, type bertipe string, properties bertipe class properties
- Class Board merupakan kelas yang berisi informasi mengenai papan permainan seperti id, width, height, features, minumum_delay_between_moves, dan game_objects. Minimum_delay_between_moves mengatur jeda waktu minimal dari jalannya bot dan game_objects berisi informasi dari objek yang terdapat dalam papan permainan. Di dalam class ini juga terdapat fungsi untuk mengecek apakah sebuah bot geraknya valid atau tidak.

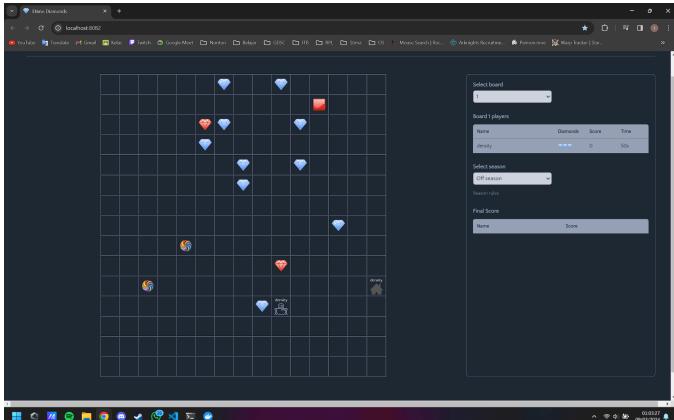
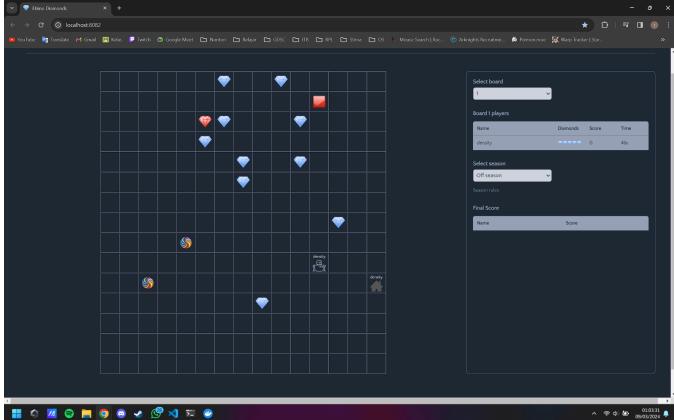
Program juga memerlukan file *util.py* agar dapat berjalan. Fungsi-fungsi yang ada di dalam file ini adalah sebagai berikut.

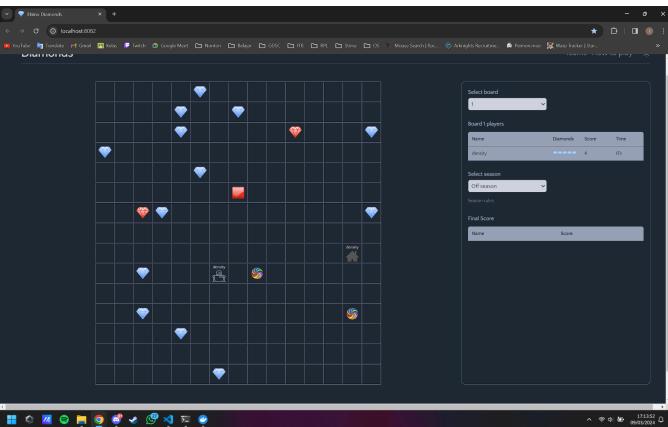
- Fungsi clamp yang memerlukan input n, smallest, dan largest akan mengembalikan nilai terbesar antara smallest dan nilai terkecil dari n atau largest.
- Fungsi get_direction yang memerlukan koordinat x dan y sebuah bot dan koordinat x dan y dari objek selain bot, fungsi ini akan mengembalikan delta_x dan delta_y sebagai arah dari gerakan bot.
- Fungsi position_equals yang memerlukan parameter position a dan b untuk membandingkan apakah position a dan b sama atau tidak.

Program juga memerlukan file-file logic yang berfungsi sebagai alternatif *greedy* dari bot. File ini berisi next_move yang mengatur cara gerak dari bot tergantung dari strategi *greedy* yang diimplementasikan, file logic ini juga mengimport fungsi get_direction dari file *util.py* agar dapat menentukan arah geraknya. Ada juga file *main.py* yang berfungsi sebagai jembatan penghubung dengan semua file dalam bot starter pack.

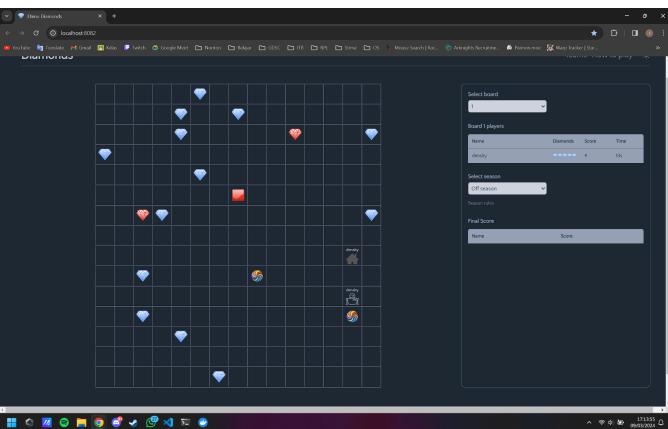
4.3. Analisis dan Pengujian

Pada bagian ini akan dilakukan pengujian terhadap beberapa fitur utama dari bot yang akan dipilih, yakni *greedy bot by highest density*

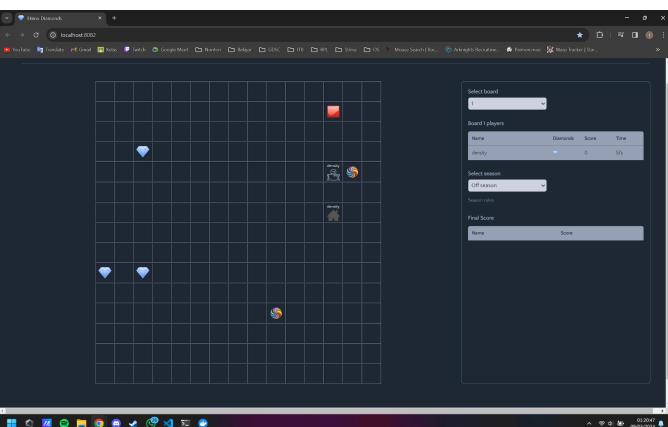
No	Gambar	Penjelasan
1.	<p>Kondisi awal</p>  <p>Kondisi sesudah</p> 	<p>Pada gambar kondisi awal, terdapat diamond biru yang jaraknya lebih dekat dengan bot tersebut. Akan tetapi bot tersebut akan mengambil diamond merah yang berjarak 2 kotak karena bot yang digunakan bertipe <i>high density</i> yang menargetkan diamond yang memiliki nilai poin per jarak yang lebih besar.</p>
2.	<p>Sebelum teleportasi</p>	<p>Gambar di samping adalah gambar yang menampilkan bot sebelum dan sesudah melakukan teleportasi untuk kembali ke base. Teleportasi dilakukan karena jarak bot menuju base lebih dekat melalui teleporter</p>



Setelah teleportasi



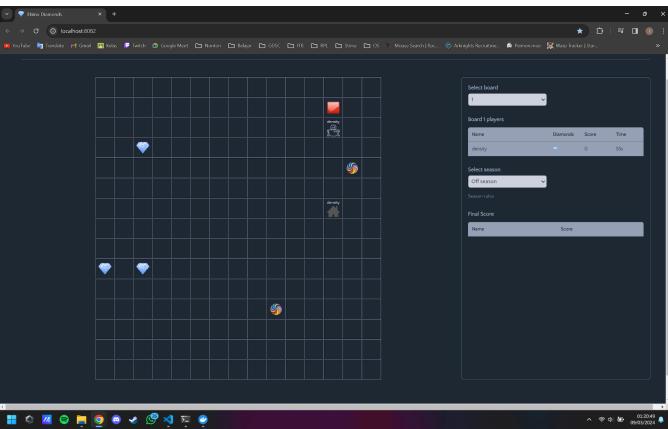
3. Kondisi awal



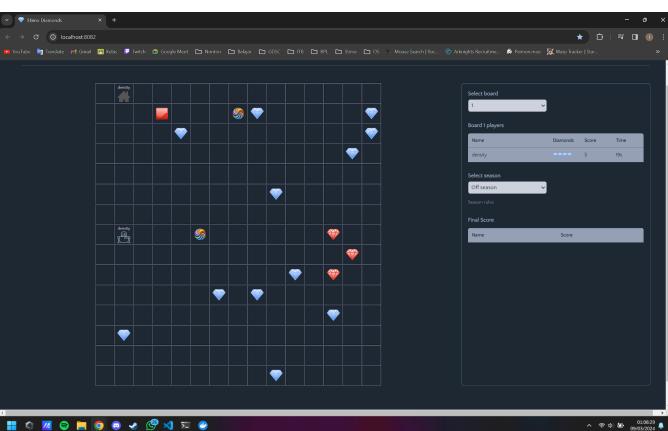
Kondisi sesudah

dibandingkan dengan hanya berjalan kaki. Dengan gambar tersebut, terbukti bahwa implementasi penggunaan teleporter berjalan dengan semestinya

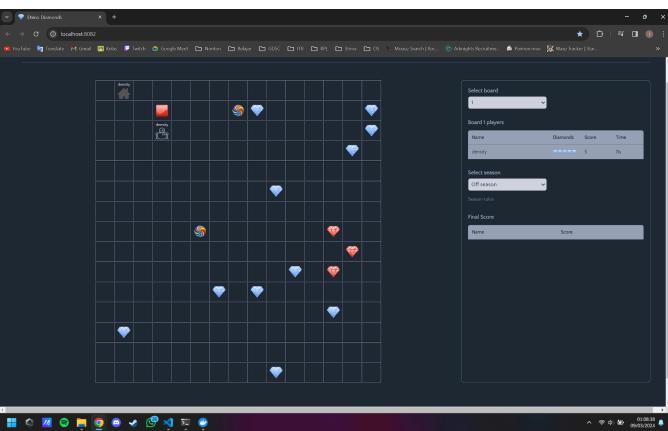
Gambar di samping memperlihatkan bot memutuskan untuk memilih diamond button karena diamond yang tersisa sudah sedikit dan jaraknya sangat jauh dari bot tersebut. Dari kedua gambar ini, implementasi untuk kondisi pemilihan red button berjalan dengan baik.



4. Kondisi awal



Kondisi sesudah



Gambar di samping menunjukkan bahwa bot akan pulang ke base ketika diamond yang dibawa telah mencapai `inventory_size - 1`. Selama perjalanan pulang, bot memeriksa diamond yang ada dan dekat di sekitar bot tersebut. Jika ada, bot akan pergi mengambil diamond tersebut terlebih dahulu dan langsung pulang setelahnya. Jika tidak ada, maka bot hanya akan pulang ke base.

BAB V

PENUTUP

5.1. Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma, kami berhasil bereksperimen untuk membuat berbagai macam bot untuk permainan Diamonds dengan strategi greedy. Bot yang kami pilih sebagai pilihan mutakhir adalah bot dengan strategi kepadatan terbesar (*highest density*). Pemilihan strategi ini didapat dari melakukan pertandingan bot dengan strategi yang berbeda-beda.

Dari pertandingan yang dilakukan, bot dengan strategi densitas terbesar berhasil membawa pulang diamond dengan rata-rata yang paling besar dibandingkan dengan bot *Highest value* dan bot *Shortest distance*.

Berdasarkan hasil pengamatan selama pertandingan, bot yang kami pilih dapat berjalan dengan cukup optimal dengan mempertimbangkan densitas diamond terbesar, penggunaan *teleporter* untuk mendapatkan rute terdekat menuju diamond, dan pergerakan bot menuju base. Hasil tadi didapat dari board dengan ukuran 15x15 dan durasi permainan selama 1 menit.

Dengan demikian, dapat disimpulkan bahwa bot dengan strategi densitas terbesar telah berhasil berjalan dengan cukup optimal dengan memperhatikan fitur yang dimiliki, perbandingan rata-rata skor akhir dari setiap bot, dan konfigurasi dari papan permainan.

5.2. Saran

Saran untuk proses pengembangan bot dalam tugas besar ini adalah sebagai berikut.

1. Implementasi kode dapat dilakukan dengan lebih rapi agar dapat memudahkan proses debugging
2. Melakukan pertandingan bot dengan kelompok lain untuk mengetahui kelebihan dan kelemahan dari bot yang telah dibuat

DAFTAR REFERENSI

Diamonds Game Engine. Diakses 2 Maret 2024.

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

Bot Starter Pack. Diakses 2 Maret 2024.

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

Munir, Rinaldi. "Algoritma Greedy (Bagian 1)." Informatika. Diakses 2 Maret 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. "Algoritma Greedy (Bagian 2)." Informatika. Diakses 2 Maret 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. "Algoritma Greedy (Bagian 3)." Informatika. Diakses 2 Maret 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).

LAMPIRAN

Link Github: https://github.com/AlbertChoe/Tubes1_terkesTima

Link Video Youtube: https://youtu.be/HilszzM_1yY