

IMPLEMENTASI ALGORITMA X DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar



Oleh: Kelompok 14

Zahwa Natasya Hamzah	123140069
Andini Rahma Kemala	123140067
Widia Salsalina Br Singarimbun	123140035

Dosen Pengampu : Winda Yulita, M.Cs.

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2025**

DAFTAR ISI

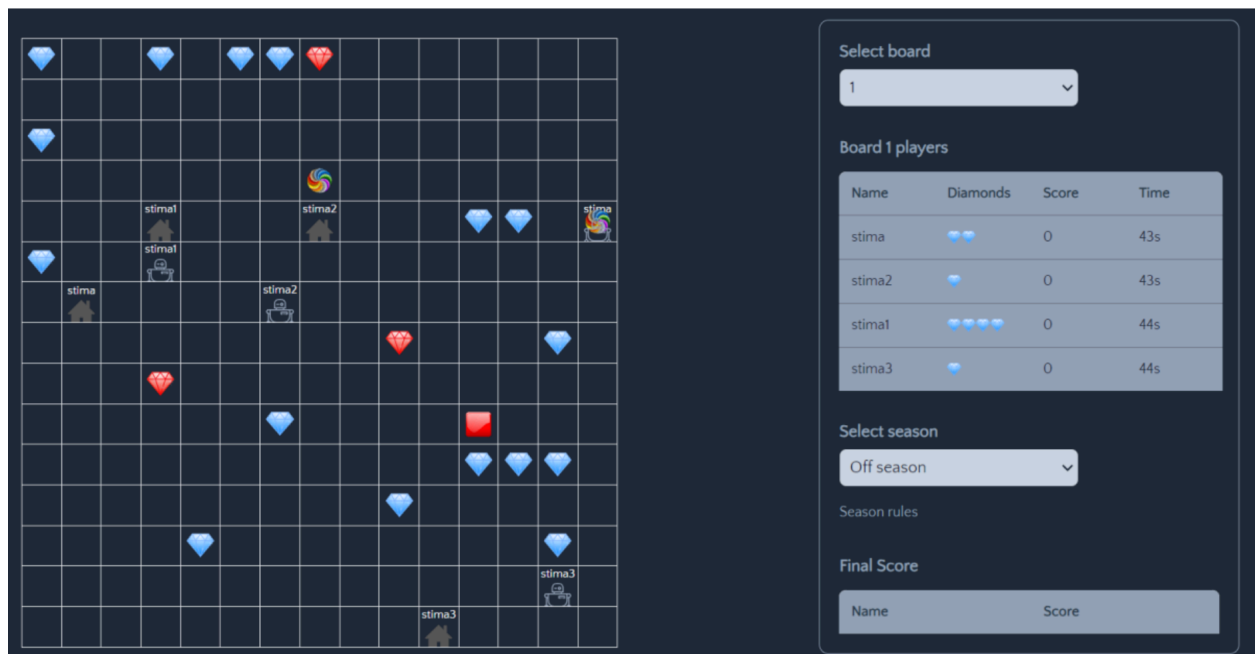
BAB I DESKRIPSI TUGAS.....	3
BAB II LANDASAN TEORI.....	7
2.1 Dasar Teori.....	7
2.2 Cara Kerja Program.....	8
2.3 Cara Implementasi Program.....	8
2. 4 Menjalankan Bot Program.....	10
2. 5 Cara mengimplementasikan bot.....	11
2. 6 Konsep greedy dalam pengambilan keputusan.....	11
2.7 Metrik Efisiensi : skor prioritas Diamond.....	12
2.8 Perhitungan jarak : Manhattan Distance.....	12
BAB III APLIKASI STRATEGI GREEDY.....	13
3.1 Proses Mapping.....	13
3.2 Eksplorasi Alternatif Solusi Greedy.....	18
3.2.1 Greedy Safe Collection.....	18
3.2.2 High-Value Diamond First.....	18
3.2.3 Trap Strategy (Pancing Musuh).....	18
3.2.4 Deny Strategy (Blok Diamond).....	19
3.2.5 Patrol Strategy.....	19
3.2.6 Explore Uncontested Area.....	19
3.2.7 Time-Aware Strategy.....	19
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy.....	19
3.3.1. Greedy Safe Collection.....	19
3.3.2. High-Value Diamond First.....	20
3.3.3. Trap Strategy (Pancing Musuh).....	20
3.3.4. Deny Strategy (Blok Diamond).....	20
3.3.5. Patrol Strategy.....	21
3.3.6. Explore Uncontested Area.....	21
3.3.7. Time-Aware Strategy.....	21
3.4 Strategi Greedy yang Dipilih.....	22
BAB IV IMPLEMENTASI DAN PENGUJIAN.....	23
4.1 Implementasi Algoritma Greedy.....	23
1. Pseudocode.....	23
2. Penjelasan Alur Program.....	27
4.2 Struktur Data yang Digunakan.....	30

4.3 Pengujian Program.....	32
4.3.1 Skenario Pengujian.....	32
4.3.2 Hasil Pengujian dan Analisis.....	35
BAB V KESIMPULAN DAN SARAN.....	39
5.1 Kesimpulan.....	39
5.2 Saran.....	39
LAMPIRAN.....	40
DAFTAR PUSTAKA.....	41

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 sederhana 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.



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:
 - 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.

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.

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

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 menempa 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

Dalam menyelesaikan persoalan pengambilan keputusan dalam permainan Diamond Collector, strategi yang digunakan oleh bot mengacu pada prinsip algoritma greedy. Algoritma greedy adalah suatu metode dalam ilmu komputer yang bertujuan untuk mencari solusi optimal dengan cara membuat keputusan terbaik di setiap langkah berdasarkan kondisi saat itu, tanpa mempertimbangkan dampak jangka panjangnya. Pendekatan ini sangat sesuai untuk permasalahan yang bersifat optimasi lokal, seperti memilih diamond terdekat dan bernilai tinggi dalam permainan ini [1].

Selain greedy, algoritma pencarian jalur seperti Breadth-First Search (BFS) juga digunakan untuk menentukan rute terpendek dari posisi bot menuju target, baik itu diamond, base, maupun musuh. BFS bekerja dengan menelusuri simpul-simpul yang berdekatan terlebih dahulu sebelum melanjutkan ke simpul yang lebih jauh, sehingga menjamin bahwa jalur yang ditemukan adalah yang paling pendek jika bobot semua lintasan sama. Dalam permainan ini, BFS berperan besar dalam menentukan efisiensi waktu tempuh bot ke lokasi strategis [2].

Untuk mendukung penilaian jarak antar objek di papan permainan, digunakan metode Manhattan Distance. Metode ini menghitung jarak antara dua titik pada grid berdasarkan jumlah langkah horizontal dan vertikal yang harus ditempuh, dengan rumus $|x_1 - x_2| + |y_1 - y_2|$. Karena permainan ini membatasi gerakan bot hanya dalam empat arah utama (atas, bawah, kiri, kanan), maka Manhattan Distance menjadi metode yang paling relevan dan akurat dalam mengukur jarak [3].

2.2 Cara Kerja Program

Permainan Diamonds dijalankan melalui beberapa tahapan utama. Pertama, engine permainan diaktifkan pada alamat tertentu, seperti localhost:8082, dan menunggu koneksi dari semua bot peserta. Jumlah bot yang dibutuhkan ditentukan melalui file konfigurasi appsettings.json. Setelah semua bot terhubung, pertandingan dimulai secara otomatis.

Selama permainan berlangsung, bot menerima data kondisi game dari engine dan merespons dengan aksi yang telah dihitung sesuai strategi. Permainan berakhir ketika waktu habis atau kondisi kemenangan tercapai. Seluruh hasil pertandingan akan dicatat dalam file .json untuk keperluan analisis dan evaluasi lebih lanjut.

2.3 Cara Implementasi Program

Cara mengimplementasikan program dengan menjalankan game engine :

a. Requirement yang harus di-install

- Node.js

[Node.js](https://nodejs.org/en) dapat diinstal pada link berikut (<https://nodejs.org/en>), berfungsi untuk menghubungkan terminal ke website

- Docker desktop

Docker desktop berfungsi untuk setup local database yang dapat diunduh pada (<https://www.docker.com/products/docker-desktop/>)

- Yarn

Yarn berfungsi sebagai penghubung, yarn dapat diinstal dengan

```
Npm install --global yarn
```

b. Instalasi dan konfigurasi awal

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

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

4. Install dependencies menggunakan Yarn

```
yarn
```

5. Setup default environment variable dengan menjalankan script berikut untuk Windows

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

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

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

Lalu jalankan script berikut untuk Windows

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

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

```
npm run build
```

- d. Jalankan perintah berikut untuk memulai game-engine

```
npm run start
```

Browser : (<http://localhost:8082/>)

Implementasi pada program menunjukkan modularitas melalui pembagian kode yang kuat. Dimana setiap fungsi memiliki tanggung jawab spesifik. Pada fungsi `bfs_path` digunakan untuk mencari jalur terpendek dari posisi bot target menggunakan BFS, yang dimana pada fungsi ini mempertimbangkan penghalang dan batas papannya. Sementara pada fungsi `reconstruct_path` digunakan untuk membangun jalur dari hasil BFS untuk menentukan langkah pergerakan. Selanjutnya ada fungsi `manhattan_distance` yang digunakan untuk menghitung jarak grid antara dua titik (gerakan horizontal/ vertikal). fungsi `diamond_priority` yang digunakan untuk mengevaluasi prioritas diamond berdasarkan formula $(jarah \times 2) - poin$, yang dimana disini jarak bobot diberikan lebih besar untuk memprioritaskan diamond terdekat. selanjutnya pada fungsi `is_enemy_nearby` untuk mendeteksi keberadaan musuh dalam radius 3 langkah yang membawa diamond dan untuk fungsi `next_move` merupakan metode utama yang mengintegrasikan semua

fungsi, yang menjalankan logika pengambilan keputusan berdasarkan status bot dan kondisi dari papan permainan. program memperhitungkan kapasitas diamond bot, keberadaan musuh dan rute mana yang efektif untuk ke diamond atau base.

2. 4 Menjalankan Bot Program

- *Requirement* yang harus di-install
 - Python (<https://www.python.org/downloads/>)
- Instalasi dan konfigurasi awal
 1. Download source code (.zip) pada ([release bot starter pack](#))
 2. Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal
 3. Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

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

4. Install dependencies menggunakan pip

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

- Jalankan bot

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

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

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

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

2. 5 Cara mengimplementasikan bot

Untuk mengimplementasikan bot, langkah pertama yang perlu dilakukan adalah membuat file Python baru di dalam folder `/game/logic/`, misalnya dengan nama `mybot.py`. Di dalam file tersebut, buatlah sebuah kelas yang mewarisi (inherit) dari kelas `BaseLogic`, lalu implementasikan konstruktor dan metode `next_move` pada kelas tersebut sesuai dengan logika bot yang diinginkan. Setelah itu, import kelas bot yang telah dibuat ke dalam file `main.py`, kemudian daftarkan ke dalam dictionary `CONTROLLERS`. Untuk menjalankan program, ikuti langkah seperti pada poin c di bagian 2, namun sesuaikan argumen `logic` pada perintah atau skrip tersebut dengan nama bot yang telah terdaftar di `CONTROLLERS`. Anda bisa menjalankan satu atau beberapa bot sekaligus menggunakan skrip `.bat` atau `.sh`.

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

2. 6 Konsep greedy dalam pengambilan keputusan

Algoritma greedy adalah pendekatan pemecahan masalah yang dimana pada setiap langkahnya memilih solusi terbaik secara lokal, dengan harapan pilihan lokal tersebut akan menghasilkan solusi yang optimal secara global tanpa memperhitungkan jangka panjangnya. Dalam bot ini implementasi greedynya terdapat pada:

- Pada pemilihan diamond : memilih diamond dengan nilai `priority` terbaik berdasarkan dengan `diamond_priority()`
- Keputusan kembali ke base: langsung kembali ke base jika inventory penuh(`diamond >= inventory_size`) , ketika diamond banyak atau ketika dalam bahaya (musuh ada di dekat)
- Penyerangan musuh: jika tidak ada diamond aman yang bisa diambil, bot akan mengejar musuh terdekat yang membawa diamond tetapi selama jaraknya masih bisa dijangkau dan itu dianggap menguntungkan

Dengan strategi ini, bot membuat keputusan cepat berdasarkan informasi saat itu, tanpa perencanaan jangka panjang

2.7 Metrik Efisiensi : skor prioritas Diamond

Bot menggunakan pendekatan greedy dengan metrik skor prioritas berbasis cost_benefit untuk menentukan atau melihat diamond mana yang paling menguntungkan. Skornya dapat dihitung dengan :

$$\text{Prioritas} = (\text{jarak} * 2) - \text{poin diamond}$$

Hal ini menjelaskan dimana jika semakin kecil skor, maka akan semakin tinggi prioritas diamond. Untuk bobot jarak ($\times 2$) yang dimana memastikan bot lebih memilih diamond terdekat, bahkan jika poinnya sedikit lebih rendah. Dan mempertimbangkan trade-off antara jarak tempuh dan nilai diamond.

2.8 Perhitungan jarak : Manhattan Distance

Untuk menghitung jarak antara objek pada papan permainan, bot menggunakan metode Manhattan Distance, dimana digunakan untuk menentukan jarak ke berlian atau musuh (yang digunakan dalam diamond_priority), menilai apakah musuh berada di dekat atau cukup untuk dianggap ancaman (digunakan dalam is_enemy_nearby) dan mengoptimalkan pergerakan bot dengan BFS. Pergerakan yang dilakukan hanya diperbolehkan dalam empat arah utama yaitu atas, bawah, kiri, kanan.

$$\text{Rumus Manhattan Distance} : |x_1 - x_2| + |y_1 - y_2|$$

Manhattan Distance disini membantu bot menilai seberapa cepat atau dekat ia dapat mencapai target tertentu, seperti diamond atau musuh. Yang dimana dalam konteks greedy, informasi jarak digunakan untuk melihat target terbaik yang paling menguntungkan secara lokal saat ini.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Dalam menerapkan algoritma greedy pada game pengambilan diamond, penting untuk terlebih dahulu mengidentifikasi dan memetakan elemen-elemen permainan ke dalam struktur dasar algoritma greedy. Langkah ini diperlukan agar strategi yang diterapkan tetap sesuai dengan kerangka kerja greedy dan mampu menghasilkan solusi yang optimal dan efisien. Berikut merupakan elemen-elemen dalam game *Diamonds* yang telah dipetakan ke dalam komponen-komponen greedy:

1. **Himpunan Kandidat (C)** mencakup seluruh objek yang dapat memengaruhi pergerakan dan keputusan bot, seperti diamond (baik merah maupun biru), posisi musuh, base sendiri, dan base musuh. Untuk Alternatif 1 dan 4, kandidat yang dipertimbangkan adalah posisi base sendiri, karena bot harus mengevaluasi kapan saat terbaik untuk kembali. Sementara itu, pada Alternatif 2 dan 5, kandidatnya adalah posisi semua diamond di peta yang masih mungkin diambil sesuai kapasitas inventory. Pada Alternatif 3, himpunan kandidat berupa posisi musuh yang membawa diamond, dan pada Alternatif 6 dan 7, kandidat bisa meluas ke semua arah yang valid untuk gerakan.
2. **Himpunan Solusi (S)** dalam konteks game ini adalah koordinat atau urutan koordinat yang dianggap paling menguntungkan jika didatangi. Misalnya, Alternatif 2 membentuk solusi berupa jalur ke diamond aman bernilai tinggi, sedangkan Alternatif 3 membentuk solusi berupa langkah-langkah menuju musuh terdekat yang membawa diamond. Pada Alternatif 1 dan 4, solusi berupa jalur balik ke base untuk menyimpan hasil, yang walaupun tidak menambah diamond, memberi nilai dalam bentuk poin aman.
3. **Fungsi Solusi** akan memverifikasi apakah kandidat menghasilkan solusi yang valid dalam kondisi spesifik, terutama menjelang akhir permainan. Misalnya, jika waktu atau langkah tersisa tidak cukup untuk mengejar musuh (Alternatif 3) atau mengambil diamond tertentu (Alternatif 2), maka fungsi ini akan mengabaikannya. Namun, dalam sebagian besar strategi seperti Alternatif 1, 2, dan 4, pemilihan solusi sudah dilakukan oleh fungsi seleksi dan kelayakan secara lebih awal.

4. **Fungsi Seleksi (Selection)** menjadi pusat utama strategi greedy. Dalam Alternatif 2, fungsi seleksi memilih diamond dengan perhitungan rasio jarak dan poin terbaik (fungsi `diamond_priority`). Dalam Alternatif 3, seleksi dilakukan berdasarkan jarak terdekat ke musuh yang membawa diamond. Sedangkan Alternatif 1 dan 4 memilih base sebagai tujuan jika kapasitas penuh atau tidak ada peluang lain. Setiap alternatif memiliki logika seleksi berbeda sesuai tujuan strateginya.
5. **Fungsi Kelayakan (Feasibility)** memverifikasi apakah kandidat bisa dijadikan solusi yang valid. Sebagai contoh, pada Alternatif 5, diamond dengan berat melebihi kapasitas inventory yang tersisa akan ditolak oleh fungsi ini. Dalam Alternatif 3, musuh yang terlalu jauh mungkin tidak feasible untuk dikejar. Fungsi ini mencegah bot melakukan aksi sia-sia yang berujung pada gerakan tidak efisien atau bug.
6. **Fungsi Objektif** digunakan untuk mengevaluasi dan membandingkan kandidat berdasarkan keuntungan maksimal atau biaya minimal. Dalam Alternatif 2, fungsi objektif mengukur trade-off antara nilai diamond dan jaraknya dari bot. Dalam Alternatif 3, objektifnya adalah mendapatkan diamond dari musuh dengan usaha seminimal mungkin. Sedangkan dalam Alternatif 1 dan 4, objektif adalah menyimpan diamond dengan aman, sehingga bot tidak kehilangan poin yang sudah didapat.

3.1.1 Alternatif Greedy by Greedy Safe Collection

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Semua diamond di papan
Himpunan Solusi (S)	Diamond yang bobotnya muat dan jauh dari musuh..
Fungsi Solusi	Ambil diamond terpilih, jalan ke arahnya..
Fungsi Seleksi (<i>Selection</i>)	Diamond dengan nilai $\text{dist} * 2 - \text{points}$ terkecil.
Fungsi Kelayakan (<i>Feasible</i>)	$\text{Bobot diamond} \leq \text{sisas kapasitas}$, dan tidak ada musuh dalam radius 2..
Fungsi Objektif	Maksimalkan jumlah diamond yang dikumpulkan dengan risiko minimum.

3.1.2 Alternatif Greedy by High-Value Diamond First

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Semua diamond di papan.
Himpunan Solusi (S)	Diamond yang muat dibawa (berat \leq sisa kapasitas).
Fungsi Solusi	Ambil diamond tersebut..
Fungsi Seleksi (<i>Selection</i>)	Pilih diamond dengan rasio points / distance tertinggi.
Fungsi Kelayakan (<i>Feasible</i>)	Diamond tidak diblokir dan bisa diakses.
Fungsi Objektif	Maksimalkan nilai (points) yang dikumpulkan per langkah..

3.1.3 Alternatif Greedy by Trap Strategy (Pancing musuh)

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Posisi musuh dalam radius 4.
Himpunan Solusi (S)	Titik posisi yang memungkinkan bot memancing musuh masuk.
Fungsi Solusi	Posisikan diri untuk menunggu/menjebak musuh.
Fungsi Seleksi (<i>Selection</i>)	Titik yang dekat dengan musuh, tapi juga dekat dengan base.
Fungsi Kelayakan (<i>Feasible</i>)	Titik bisa dicapai dan cukup aman untuk sementara..
Fungsi Objektif	Menjatuhkan musuh yang membawa diamond.

3.1.4 Alternatif Greedy by Deny Strategy (Blok Diamond)

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Diamond di sekitar musuh.
Himpunan Solusi (S)	Diamond yang bisa diakses sebelum musuh.
Fungsi Solusi	Bot berdiri di atas diamond tersebut (jangan diambil).
Fungsi Seleksi (<i>Selection</i>)	Diamond terdekat dari musuh tapi juga terjangkau oleh bot.
Fungsi Kelayakan (<i>Feasible</i>)	Diamond tidak terlalu dekat musuh, bisa diambil lebih dulu.
Fungsi Objektif	Mencegah musuh mendapatkan diamond.

3.1.5 Alternatif Greedy by Patrol Strategy

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Titik-titik jalur lalu lintas musuh (prediksi / tengah map).
Himpunan Solusi (S)	Titik-titik yang meng-cover lebih dari satu jalur.
Fungsi Solusi	Bot berpatroli ke titik-titik tersebut secara berkala.
Fungsi Seleksi (<i>Selection</i>)	Titik dengan frekuensi musuh muncul tertinggi.
Fungsi Kelayakan (<i>Feasible</i>)	Titik bisa dicapai dan aman untuk beberapa langkah.
Fungsi Objektif	Tingkatkan peluang mencegah musuh sambil jaga posisi.

3.1.6 Alternatif Greedy by Explore Uncontested Area

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Semua petak yang belum dikunjungi.
Himpunan Solusi (S)	Petak kosong yang tidak berisi dinding/musuh.
Fungsi Solusi	Bot bergerak untuk menjelajahi area kosong.
Fungsi Seleksi (<i>Selection</i>)	Petak terdekat dengan diamond tersembunyi potensial..
Fungsi Kelayakan (<i>Feasible</i>)	Petak bisa dicapai dan tidak ada musuh dekat.
Fungsi Objektif	Temukan diamond tersembunyi / posisi strategis baru.

3.1.7 Alternatif Greedy by Greedy Safe Collection

Elemen Algoritma Greedy	Penjelasan
Himpunan Kandidat (C)	Posisi base dan diamond yang mungkin bisa diambil sebelum waktu habis.
Himpunan Solusi (S)	Pulang ke base atau cari diamond terdekat jika masih cukup waktu.
Fungsi Solusi	Lari ke base tanpa risiko.
Fungsi Seleksi (<i>Selection</i>)	Jika waktu tersisa $<$ threshold \rightarrow pulang.
Fungsi Kelayakan (<i>Feasible</i>)	Bisa sampai ke base tepat waktu.
Fungsi Objektif	Hindari kehilangan diamond saat waktu habis.

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam game *Diamonds*, setiap bot saling bersaing untuk mengumpulkan diamond terbanyak dalam waktu yang terbatas. Untuk itu, digunakan pendekatan **algoritma greedy**, yaitu strategi yang selalu memilih tindakan paling menguntungkan di setiap langkah permainan. Dalam eksplorasi ini, dikembangkan tujuh strategi greedy yang masing-masing dirancang untuk menghadapi kondisi permainan yang berbeda. Setiap strategi mengevaluasi posisi diamond, pergerakan musuh, waktu, serta kapasitas penyimpanan diamond. Berikut penjelasan eksplorasi dari masing-masing strategi:

3.2.1 Greedy Safe Collection

Strategi ini fokus pada pengumpulan diamond yang aman. Bot akan memilih diamond yang tidak hanya bernilai tinggi tetapi juga memiliki risiko kecil, yaitu tidak dikelilingi musuh dan masih muat dalam inventaris. Diamond yang terlalu dekat dengan musuh akan dihindari, dan bot akan lebih memilih target yang bisa diambil dengan aman. Pendekatan ini cocok digunakan di awal permainan saat ingin mengumpulkan poin stabil tanpa gangguan.

3.2.2 High-Value Diamond First

Dalam strategi ini, bot mengejar diamond yang memberikan nilai tertinggi dibandingkan dengan jarak tempuhnya. Perhitungan dilakukan berdasarkan rasio *nilai / jarak*, sehingga bot akan mengutamakan diamond yang bernilai besar tetapi tetap terjangkau. Strategi ini sangat efektif di area permainan yang relatif aman, karena mengejar diamond bernilai tinggi di area berisiko bisa menjadi bumerang jika tidak mempertimbangkan musuh.

3.2.3 Trap Strategy (Pancing Musuh)

Alih-alih berburu diamond, strategi ini menjadikan musuh sebagai target. Bot akan memosisikan dirinya di area dekat base musuh atau tempat strategis untuk memancing musuh yang membawa diamond. Dengan memanfaatkan kemampuan *tackle*, bot bisa menjatuhkan musuh dan mengambil diamond mereka. Strategi ini sangat cocok di pertengahan atau akhir permainan saat musuh mulai membawa banyak diamond dan risiko mengambil sendiri terlalu tinggi.

3.2.4 Deny Strategy (Blok Diamond)

Strategi deny berfungsi untuk mencegah musuh mendapatkan diamond. Bot akan bergerak ke arah diamond yang berpotensi diambil oleh musuh dan mencoba berdiri di atasnya tanpa mengambilnya, sehingga musuh tidak bisa mengaksesnya. Ini adalah strategi sabotase yang ampuh, terutama saat musuh sedang unggul dalam pengumpulan poin. Strategi ini lebih bersifat defensif dan bertujuan menahan laju skor lawan.

3.2.5 Patrol Strategy

Strategi ini membuat bot melakukan patroli di area lalu lintas musuh. Dengan memantau titik-titik strategis seperti pusat peta atau jalur umum, bot meningkatkan kemungkinan untuk mencegah musuh atau memanfaatkan diamond yang dilewatkan oleh mereka. Pendekatan ini menyeimbangkan antara posisi bertahan dan menyerang, dan cocok untuk mempertahankan kontrol wilayah serta mengganggu mobilitas musuh.

3.2.6 Explore Uncontested Area

Di strategi ini, bot akan fokus menjelajahi area kosong yang belum terjamah oleh bot lain. Tujuannya adalah menemukan diamond tersembunyi yang mungkin belum terlihat di awal permainan atau tidak diperebutkan. Bot menghindari wilayah ramai yang berisiko tinggi dan mencari celah di zona sepi yang tetap menguntungkan. Strategi ini sangat berguna untuk eksplorasi awal dan memperluas kendali wilayah.

3.2.7 Time-Aware Strategy

Strategi ini memperhatikan sisa waktu permainan. Jika waktu hampir habis, bot akan langsung kembali ke base untuk menyimpan diamond yang sudah dikumpulkan. Sebaliknya, jika waktu masih cukup, bot bisa mengejar diamond terdekat. Strategi ini menghindari kerugian besar akibat kehilangan diamond saat permainan selesai, dan sangat penting di fase akhir pertandingan ketika setiap poin sangat berarti.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

3.3.1. Greedy Safe Collection

Efisiensi:

Strategi ini efisien karena hanya mempertimbangkan diamond yang aman dan dekat. Fungsi seleksinya sederhana ($\text{jarak} * 2 - \text{nilai}$), dan pemrosesan dilakukan pada subset kecil dari semua diamond (yang jauh dari musuh dan muat di inventory). Ini mempercepat pengambilan keputusan secara signifikan.

Efektivitas:

Sangat efektif dalam permainan dengan banyak musuh agresif. Fokusnya pada keamanan menjadikan strategi ini ideal untuk mempertahankan poin. Namun, potensi pengumpulan nilai bisa lebih rendah karena menghindari diamond yang berisiko meskipun bernilai tinggi.

3.3.2. High-Value Diamond First

Efisiensi:

Strategi ini tetap efisien karena hanya memerlukan evaluasi rasio points/distance untuk setiap diamond. Operasi ini ringan dan cepat dihitung.

Efektivitas:

Sangat efektif untuk memperoleh poin tertinggi dalam waktu singkat. Namun, karena tidak mempertimbangkan risiko, bot bisa menjadi sasaran empuk jika mengambil diamond yang terlalu jauh atau dekat dengan musuh.

3.3.3. Trap Strategy (Pancing Musuh)

Efisiensi:

Tidak terlalu berat secara komputasi karena hanya melihat musuh dalam radius tertentu dan menghitung posisi jebakan (biasanya sedikit titik kandidat). Strategi ini dapat diterapkan secara berkala tanpa membebani sistem.

Efektivitas:

Efektif dalam permainan melawan bot yang agresif dan sering membawa diamond. Dapat menciptakan *comeback moment*. Namun, tidak efektif jika musuh pasif atau tidak membawa diamond.

3.3.4. Deny Strategy (Blok Diamond)

Efisiensi:

Sedikit lebih mahal dari strategi biasa karena harus membandingkan posisi bot dan musuh untuk setiap diamond. Namun, masih tergolong ringan karena fokusnya terbatas pada diamond di sekitar musuh.

Efektivitas:

Sangat efektif untuk menghentikan akumulasi skor musuh dan mengontrol permainan. Tetapi karena bot hanya memblokir dan tidak mengambil, skor bot sendiri mungkin tidak langsung meningkat.

3.3.5. Patrol Strategy

Efisiensi:

Strategi ini bergantung pada prediksi dan statistik pergerakan musuh. Jika dilakukan sederhana (misalnya hanya di tengah map atau titik umum), maka tetap efisien. Namun, penggunaan data historis atau prediksi kompleks bisa memperlambat proses.

Efektivitas:

Efektif untuk menjaga wilayah dan mengganggu pergerakan musuh. Cocok untuk memperkuat kontrol area, tapi kurang efektif jika fokus utamanya adalah akumulasi diamond cepat.

3.3.6. Explore Uncontested Area

Efisiensi:

Strategi ini memiliki efisiensi sedang. Bot harus memindai area yang belum dikunjungi, yang bisa menjadi mahal jika peta besar. Namun, dapat dibatasi dengan radius eksplorasi untuk menjaga efisiensi.

Efektivitas:

Efektif untuk menemukan diamond tersembunyi atau area yang belum tersentuh musuh. Sangat berguna saat permainan mulai stagnan atau jika semua pemain fokus di satu area.

3.3.7. Time-Aware Strategy

Efisiensi:

Sangat efisien karena hanya dipicu saat waktu hampir habis. Cukup mengecek waktu dan estimasi jarak ke base, lalu memutuskan pulang atau mengambil diamond terdekat.

Efektivitas:

Sangat penting untuk mempertahankan poin yang telah dikumpulkan. Menjamin bahwa upaya sepanjang permainan tidak sia-sia. Tidak meningkatkan skor secara langsung, tapi mencegah kehilangan skor besar di akhir.

3.4 Strategi Greedy yang Dipilih

Berdasarkan penjabaran beberapa alternatif strategi bot greedy di atas, kami memutuskan untuk menggunakan pendekatan **Alternatif Greedy All In Diamond**. Pilihan ini kami ambil karena pendekatan tersebut merupakan gabungan dari berbagai strategi greedy yang telah dijelaskan sebelumnya, sehingga memiliki cakupan pertimbangan yang lebih luas dan fleksibel. Selain itu, pendekatan ini juga secara eksplisit **mengimplementasikan dua strategi utama**, yaitu **Greedy Safe Collection** dan **Trap Strategy**, yang telah terbukti memberikan hasil yang konsisten dan adaptif pada berbagai kondisi permainan.

Kami menilai bahwa pendekatan ini memiliki keunggulan dalam konsistensi performa karena mampu menyesuaikan diri dengan situasi permainan — misalnya, mengutamakan pengambilan diamond yang aman saat situasi mendukung, atau beralih ke pengejaran musuh saat opsi pengumpulan tidak tersedia. Hal ini membuat strategi All In Diamond terlihat lebih stabil dan tangguh dibanding strategi yang hanya fokus pada satu aspek.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
import random
from typing import Optional, List, Tuple
from collections import deque

from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position

class Gembot(BaseLogic):
    def _init_(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]

    def bfs_path(self, board: Board, start: Position, target: Position) ->
Optional[List[Tuple[int, int]]]:
        visited = set()
        queue = deque([(start.x, start.y)])
        parents = {}

        while queue:
            x, y = queue.popleft()
            if (x, y) in visited:
                continue
            visited.add((x, y))

            if (x, y) == (target.x, target.y):
                return self.reconstruct_path(parents, (x, y), (start.x, start.y))

            for dx, dy in self.directions:
                nx, ny = x + dx, y + dy
                if not (0 <= nx < board.width and 0 <= ny < board.height):
                    continue
```



```

        if any(obj.position.x == nx and obj.position.y == ny and obj.type
== "wall" for obj in board.game_objects):
            continue
        if (nx, ny) not in parents:
            parents[(nx, ny)] = (x, y)
            queue.append((nx, ny))
    return None

def reconstruct_path(self, parents, goal, start):
    path = []
    current = goal
    while current != start:
        path.append(current)
        current = parents[current]
    path.reverse()
    return path

def manhattan_distance(self, pos1: Position, pos2: Tuple[int, int]) -> int:
    return abs(pos1.x - pos2[0]) + abs(pos1.y - pos2[1])

def diamond_priority(self, diamond: GameObject, bot_pos: Position) -> int:
    dist = self.manhattan_distance(bot_pos, (diamond.position.x,
diamond.position.y))
    points = getattr(diamond.properties, 'points', 1)
    return dist * 2 - points

def is_enemy_nearby(self, board: Board, bot_pos: Position, radius: int = 3)
-> bool:
    for enemy in board.bots:
        if enemy.position != bot_pos and enemy.properties.diamonds > 0:
            dist = self.manhattan_distance(bot_pos, (enemy.position.x,
enemy.position.y))
            if dist <= radius:
                return True
    return False

def next_move(self, board_bot: GameObject, board: Board) -> Tuple[int, int]:
    props = board_bot.properties
    current_pos = board_bot.position

```

```

my_x, my_y = current_pos.x, current_pos.y

target = None

# Kembali ke base jika penuh atau dalam bahaya
if props.diamonds >= props.inventory_size or (
    props.diamonds >= max(2, props.inventory_size - 1) and
self.is_enemy_nearby(board, current_pos, radius=3)):
    target = props.base

# Prioritas diamond aman dan bisa diambil
if not target:
    sisa = props.inventory_size - props.diamonds
    diamonds = [
        d for d in board.game_objects
        if d.type == "DiamondGameObject" and getattr(d.properties,
'weight', 1) <= sisa
    ]
    safe_diamonds = []
    for d in diamonds:
        if all(self.manhattan_distance(Position(d.position.x,
d.position.y), (e.position.x, e.position.y)) > 2
            for e in board.bots if e != board_bot):
            safe_diamonds.append(d)

    if safe_diamonds:
        safe_diamonds.sort(key=lambda d: self.diamond_priority(d,
current_pos))
        target = safe_diamonds[0].position

# Jika tidak ada diamond aman, cegat musuh terdekat yang bawa diamond
if not target:
    enemies = [e for e in board.bots if e != board_bot and
e.properties.diamonds > 0]
    if enemies:
        enemies.sort(key=lambda e: self.manhattan_distance(current_pos,
(e.position.x, e.position.y)))
        nearest_enemy = enemies[0]

```

```

        if self.manhattan_distance(current_pos,
(nearest_enemy.position.x, nearest_enemy.position.y)) <= 3:
            target = nearest_enemy.position

# Jika tidak ada prioritas lain, pulang
if not target:
    target = props.base

# Gerak menuju target dengan BFS
path = self.bfs_path(board, current_pos, target)
if path and len(path) > 0:
    next_pos = path[0]
    dx = next_pos[0] - my_x
    dy = next_pos[1] - my_y
    if board.is_valid_move(current_pos, dx, dy):
        return dx, dy

# Fallback move
for dx, dy in self.directions:
    nx, ny = my_x + dx, my_y + dy
    if board.is_valid_move(current_pos, dx, dy):
        if not any(obj.position.x == nx and obj.position.y == ny and
obj.type == "wall"
                    for obj in board.game_objects):
            return dx, dy

return 0, -1

```

2. Penjelasan Alur Program

Berikut merupakan implementasi algoritma Greedy yang dilakukan bot dalam bentuk python. Strategi diimplementasikan dalam file [mybot.py](#)

Kamus dalam [mybot.py](#)

```
import random
from typing import Optional, List, Tuple
from collections import deque
from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
```

```
class Gembot(BaseLogic):
    def __init__(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
```

__init__ method :

- direction : 4 arah gerakan (kanan, kiri, atas, bawah)

```
def reconstruct_path(self, parents, goal, start):
    path = []
    current = goal
    while current != start:
        path.append(current)
        current = parents[current]
    path.reverse()
    return path
```

- Reconstruct_path berfungsi untuk membangun kembali jalur dari goal ke start berdasarkan directory parents.

```
def manhattan_distance(self, pos1: Position, pos2: Tuple[int, int]) -> int:
    return abs(pos1.x - pos2[0]) + abs(pos1.y - pos2[1])
```

- Fungsi Manhattan_Distance() untuk menghitung jarak antara dua titik pada grid atau peta permainan dengan menggunakan $\text{jarak} = |x_1 - x_2| + |y_1 - y_2|$

```
def diamond_priority(self, diamond: GameObject, bot_pos: Position) -> int:
    dist = self.manhattan_distance(bot_pos, (diamond.position.x,
diamond.position.y))
    points = getattr(diamond.properties, 'points', 1)
    return dist * 2 - points
```

Fungsi `diamond_priority()` melakukan hal ini dengan:

- Mengukur jarak ke diamond menggunakan `Manhattan_Distance`
- Mengambil nilai poin diamond tersebut
- Menghitung skor prioritas dengan formula: skor prioritas = (jarak x 2) - poin
- Skor ini akan digunakan untuk mengurutkan diamond dan memilih yang paling menguntungkan secara lokal.

```
if props.diamonds >= props.inventory_size or (
    props.diamonds >= max(2, props.inventory_size - 1) and
self.is_enemy_nearby(board, current_pos, radius=3)):
    target = props.base
```

- `props.diamonds >= max(2, props.inventory_size - 1)` fungsinya untuk mengecek diamond yang dimiliki bot sudah cukup banyak, menjadi batas agar bot dapat mempertimbangkan untuk kembali jika kondisi darurat muncul, walaupun belum benar-benar penuh.
- `self.is_enemy_nearby(board, current_pos, radius=3)` fungsinya jika musuh yang mendekat dengan diamond, bot harus berhati-hati dan mempertimbangkan untuk pulang agar diamond tidak hilang jika diserang.
- `target = props.base` fungsinya untuk bot diarahkan pulang untuk mengamankan hasil diamond yang sudah penuh dikumpulkan.

```
if not target:
    sisa = props.inventory_size - props.diamonds
    diamonds = [
        d for d in board.game_objects
        if d.type == "DiamondGameObject" and getattr(d.properties,
'weight', 1) <= sisa
```

```

    ]
    safe_diamonds = []
    for d in diamonds:
        if all(self.manhattan_distance(Position(d.position.x,
d.position.y), (e.position.x, e.position.y)) > 2
            for e in board.bots if e != board_bot):
            safe_diamonds.append(d)

    if safe_diamonds:
        safe_diamonds.sort(key=lambda d: self.diamond_priority(d,
current_pos))
        target = safe_diamonds[0].position

```

- if not target: fungsinya untuk mengecek apakah belum ada target yang ditentukan
- sisa = props.inventory_size - props.diamonds fungsinya untuk menghitung kapasitas inventory yang masih kosong
- diamonds = [.. fungsinya untuk memfilter diamond yang jaraknya lebih dari 2 dari semua musuh
- if safe_diamonds: fungsinya untuk mengurutkan diamond aman berdasarkan nilai terbaik, lalu set diamond dengan nilai tertinggi sesuai target.

```

if not target:
    enemies = [e for e in board.bots if e != board_bot and
e.properties.diamonds > 0]
    if enemies:
        enemies.sort(key=lambda e: self.manhattan_distance(current_pos,
(e.position.x, e.position.y)))
        nearest_enemy = enemies[0]
        if self.manhattan_distance(current_pos,
(nearest_enemy.position.x, nearest_enemy.position.y)) <= 3:
            target = nearest_enemy.position

```

- if not target: berfungsi untuk mengecek apakah belum ada target yang ditentukan sebelumnya
- enemies = .. berfungsi untuk membuat list musuh yang berbeda dari bot sendiri dan sedang membawa diamond
- if enemies: berfungsi jika ada musuh yang memenuhi kriteria

- ## 4.2 Struktur Data yang Digunakan

```
src/
|
|----game/
|
|    |--__pycache__/
|
|    |--logic/
|    |
|    |    |--__init__.py
|    |
|    |    |--base.py
|    |
|    |    |--ctaboy.py
|    |
|    |    |--mybot.py
|    |
|    |    |--random.py
|
|
|    |--__init__.py
|
|    |--api.py
```

```
|      |----- board_handler.py
|      |
|      |----- bot_handler.py
|      |
|      |----- models.py
|      |
|      |----- util.py
|
|----- .gitignore
|
|----- decode.py
|
|----- main.py
|
|----- README.md
|
|----- requirements.txt
|
|----- run.bots.bat
|
|----- run-bots.sh
```

Proyek ini menggunakan format modular yang terbagi menjadi tiga elemen kunci. Folder `game/` menyimpan modul dasar permainan yang terdiri dari model data, fungsi-fungsi utilitas, dan pengelola untuk manajemen permainan. Subfolder `logic/` secara khusus berisi realisasi kecerdasan buatan bot yang mengusung strategi utama Greedy yang terdapat dalam `mybot.py`, serta didukung oleh beberapa variasi bot lainnya. Di level tertinggi, file-file seperti `main.py` berfungsi sebagai titik masuk, `requirements.txt` mengatur ketergantungan, dan skrip eksekusi berfungsi sebagai pengendali untuk operasi. Struktur ini sengaja diatur untuk secara jelas memisahkan logika permainan, algoritma bot, dan sistem pendukung, sehingga proses pengembangan dan pemeliharaan menjadi lebih mudah.

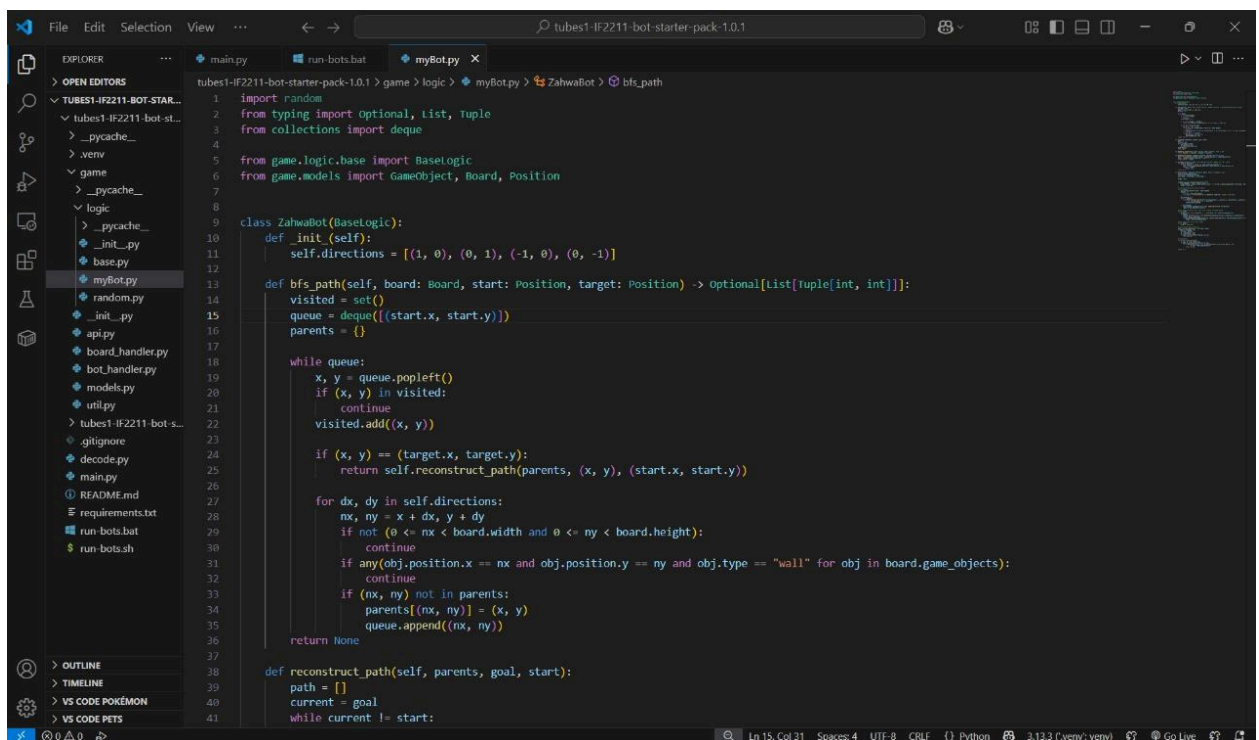
4.3 Pengujian Program

4.3.1 Skenario Pengujian

Bot ZahwaBot diuji di lingkungan lokal menggunakan Docker container yang terhubung dengan game engine Etimo lewat protokol HTTP. Saat game berjalan, engine mengirim data kondisi permainan ke endpoint bot untuk menentukan langkah berikutnya. Bot memproses data tersebut dengan algoritma pencarian jalur dan strategi pengambilan diamond yang aman dan efisien, lalu mengirimkan arah gerakan sebagai respons. Pengujian menggunakan container memastikan lingkungan yang konsisten, mudah dipindahkan, dan mendukung pengujian otomatis serta paralel.

Langkah-langkah pengujian:

1. Siapkan bot yang telah di buat.



```
1 import random
2 from typing import Optional, List, Tuple
3 from collections import deque
4
5 from game.logic.base import BaseLogic
6 from game.models import GameObject, Board, Position
7
8
9 class ZahwaBot(BaseLogic):
10     def __init__(self):
11         self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
12
13     def bfs_path(self, board: Board, start: Position, target: Position) -> Optional[List[Tuple[int, int]]]:
14         visited = set()
15         queue = deque([(start.x, start.y)])
16         parents = {}
17
18         while queue:
19             x, y = queue.popleft()
20             if (x, y) in visited:
21                 continue
22             visited.add((x, y))
23
24             if (x, y) == (target.x, target.y):
25                 return self.reconstruct_path(parents, (x, y), (start.x, start.y))
26
27             for dx, dy in self.directions:
28                 nx, ny = x + dx, y + dy
29                 if not (0 <= nx < board.width and 0 <= ny < board.height):
30                     continue
31                 if any(obj.position.x == nx and obj.position.y == ny and obj.type == "wall" for obj in board.game_objects):
32                     continue
33                 if (nx, ny) not in parents:
34                     parents[(nx, ny)] = (x, y)
35                 queue.append((nx, ny))
36
37         return None
38
39     def reconstruct_path(self, parents, goal, start):
40         path = []
41         current = goal
42         while current != start:
```

Pada program ini kami menggunakan bahasa python untuk membuat bot Etimo game.

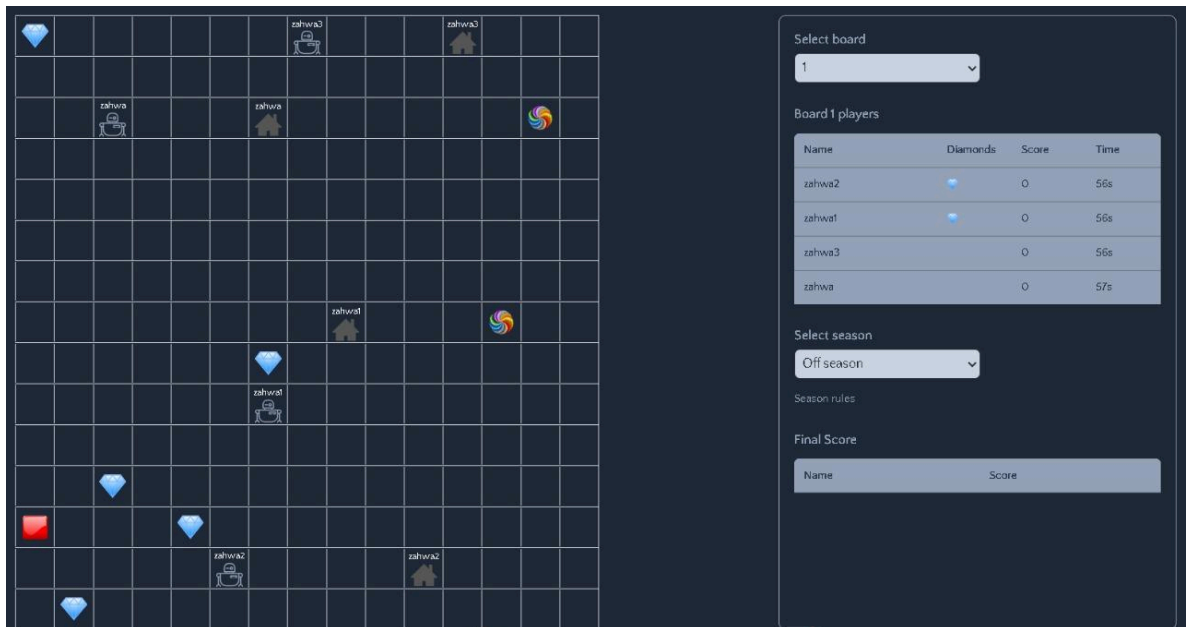
2. Run bot melalui Terminal/powershell menggunakan ./run-bots.bat

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\muham\Downloads\tubes1-IF2211-bot-starter-pack-1.0.1>cd C:\Users\muham\Downloads\tubes1-IF2211-game-engine-1.1.0
C:\Users\muham\Downloads\tubes1-IF2211-game-engine-1.1.0>npm run start
```

Pengujian dijalankan melalui Terminal/PowerShell dengan mengeksekusi file `run-bots.bat` (terdapat typo pada gambar yang menulis "`./run-bots.but`"). Ini merupakan script batch Windows untuk menjalankan bot secara otomatis.

3. Etime bot berhasil dijalankan



Skenario pengujian etimo bot:

- Skenario pertama: test kemampuan pencarian clustering

Menguji kemampuan ZahwaBot dalam mengenali dan memprioritaskan area pada peta yang memiliki konsentrasi (clustering) diamond sebagai target utama pencarian. Yang dimana peta permainan diatur sehingga terdapat beberapa area dengan jumlah diamond yang berbeda-beda, termasuk satu area yang memiliki kumpulan diamond dalam jumlah besar (cluster), yang dimana bot diharapkan mampu mendeteksi area tersebut meskipun jaraknya lebih jauh dari posisi awal.

Test casenya: bot ditempatkan di tengah peta, dan cluster diamond ditempatkan di pojok kanan atas. Terdapat juga beberapa diamond tunggal tersebar dekat dengan

posisi bot. Bot seharusnya mengabaikan diamond tunggal dan bergerak menuju cluster.

Indikator keberhasilan:

- Bot memprioritaskan blok dengan total nilai diamond tertinggi
- Bot menghindari wilayah bernilai rendah meskipun dekat
- Bot menunjukkan perilaku konsisten dalam berbagai konfigurasi peta

- Skenario kedua: test kemampuan kembali ke base

Tujuannya dengan menguji kemampuan ZahwaBot untuk mengenali kondisi saat harus kembali ke base, memilih jalur optimal, serta menghindari risiko musuh yang dapat mencuri diamond. Yang dimana bot diberikan kapasitas maksimal 5 diamond. Setelah mengumpulkan 5 diamond atau saat mengumpulkan 3-4 diamond dan musuh berada di sekitar radius 3 sel, bot seharusnya memutuskan untuk kembali ke base.

Test casenya: bot diletakkan dengan jarak 6 langkah dari base. Setelah berhasil mengumpulkan 5 diamond, bot harus memulai pergerakan pulang dengan memanfaatkan BFS untuk jalur tercepat. Terdapat juga musuh di dekat jalur alternatif, dan bot diharapkan memilih jalur aman.

Indikator keberhasilan:

- Bot mengenali kondisi penuh atau kondisi bahaya untuk kembali
- Bot memilih jalur tercepat dan teraman
- Bot secara konsisten kembali ke base di berbagai konfigurasi ancaman dan distribusi diamond.

- Skenario ketiga : test prioritas area diamond terdekat

Tujuannya untuk Menguji kemampuan ZahwaBot dalam memprioritaskan area dengan konsentrasi diamond tinggi yang berada dalam radius dekat dari posisi bot. yang dimana Diletakkan dua cluster diamond: satu di radius 2 sel dari bot dan satu lagi di jarak 5 sel. Cluster dekat berisi diamond dengan nilai cukup, sedangkan yang jauh bernilai lebih tinggi.

Test case: Bot berada di tengah peta. Di radius 2 sel, terdapat 3 diamond masing-masing bernilai 2 poin. Di jarak 5 sel, ada 5 diamond bernilai 2 poin. Bot

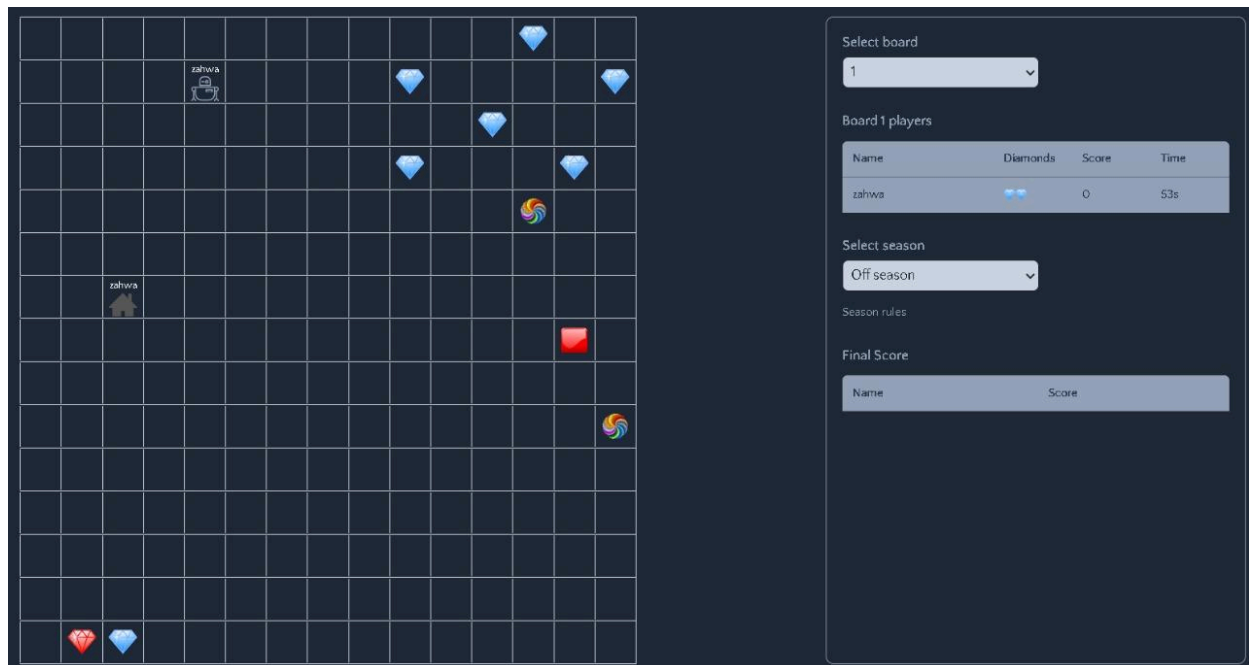
seharusnya memilih cluster yang dekat terlebih dahulu untuk efisiensi waktu, sebelum bergerak ke cluster yang lebih jauh.

Indikator keberhasilan:

- Bot memprioritaskan area dekat yang padat.
 - Bot mengabaikan area jauh jika area dekat cukup bernilai.
 - Bot menunjukkan adaptasi terhadap perubahan distribusi diamond pada peta.
- Skenario keempat :

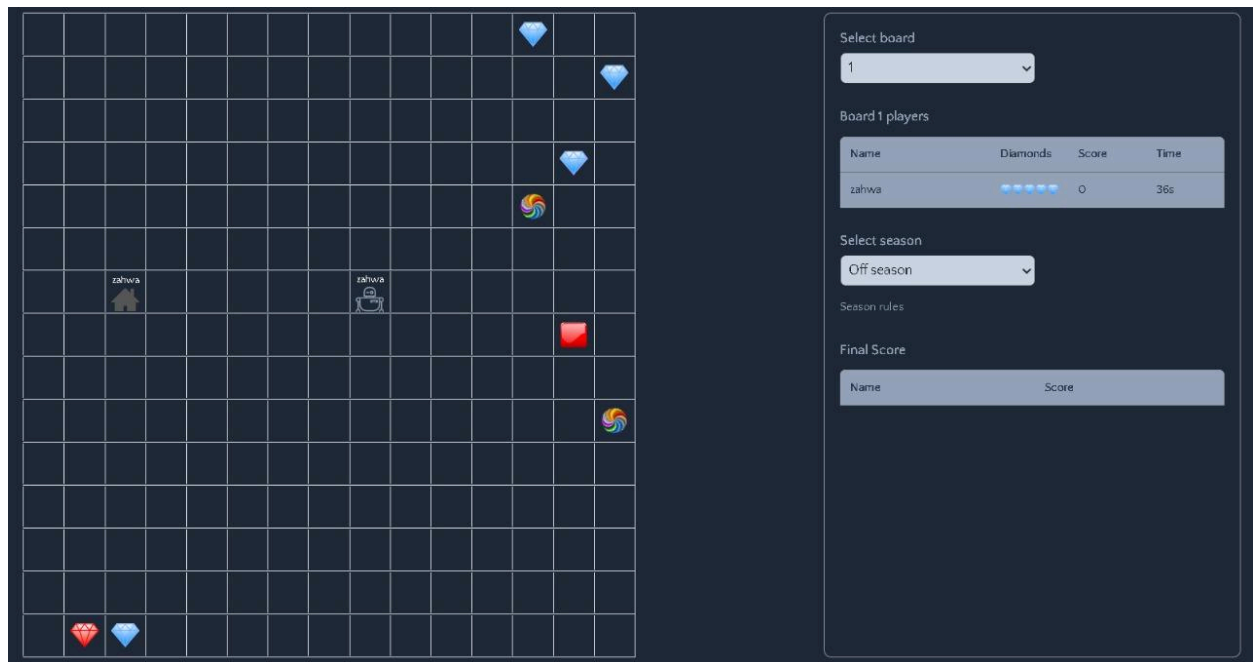
4.3.2 Hasil Pengujian dan Analisis

a. Tes kemampuan pencarian clustering



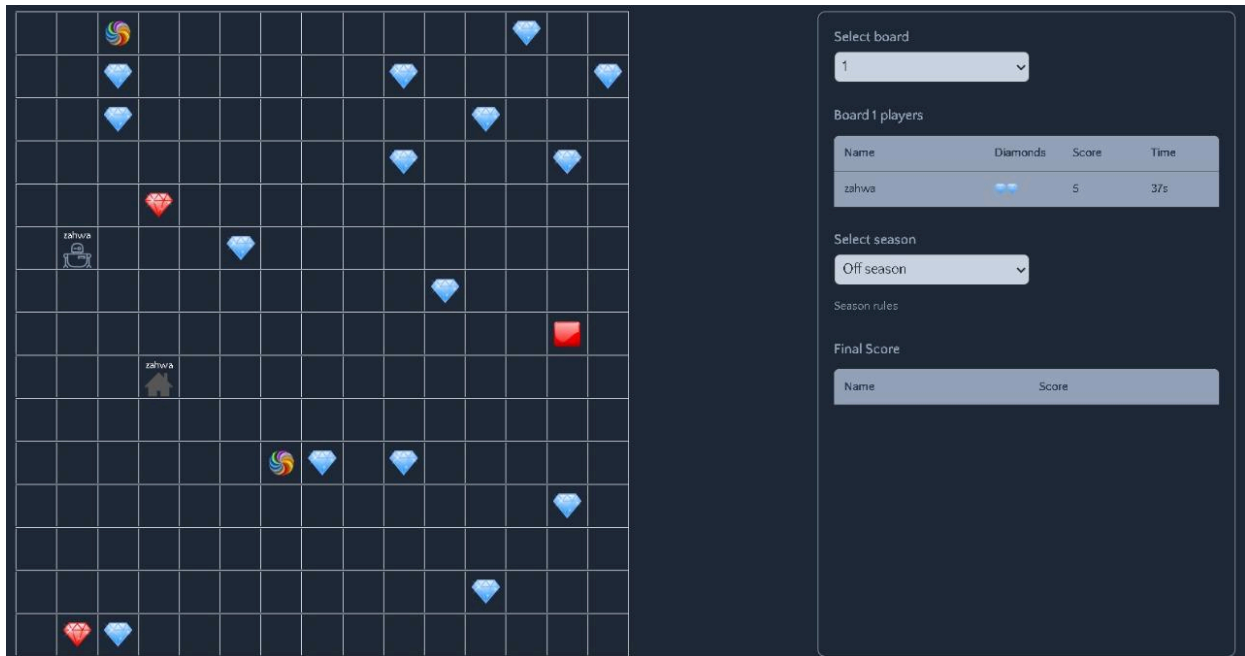
Gambar tersebut menunjukkan pengujian bot yang menggunakan metode pengelompokan area 3x3 pada peta game untuk mengumpulkan diamond. Bot menganalisis total diamond di tiap bagian, memilih bergerak ke area dengan nilai diamond tertinggi, bukan hanya berdasarkan jarak terdekat. Strategi ini bertujuan agar bot lebih efisien dalam mengumpulkan diamond, meningkatkan skor secara keseluruhan dalam waktu yang lebih singkat.

b. Tes kemampuan kembali ke base



Gambar menunjukkan pengujian kemampuan bot untuk kembali ke markas setelah mengumpulkan diamond. Bot berhasil mengumpulkan 5 diamond dan kembali dalam 30 detik, menandakan algoritma mampu mengatur waktu, rute, dan strategi penyimpanan skor dengan efisien.

- c. Tes kemampuan fokus ke daerah dengan densitas diamond tertinggi jika jarak dekat



Gambar menunjukkan pengujian kemampuan robot untuk fokus pada area dengan kepadatan diamond tinggi yang berada dekat. Robot berhasil memilih area dengan konsentrasi diamond terbesar di tengah peta dan mengumpulkan 9 diamond dalam 37 detik. Hasil ini membuktikan bahwa algoritma robot efektif dalam memprioritaskan area bernilai tinggi berdasarkan jarak dan kepadatan, sehingga mampu memaksimalkan skor secara efisien.

[illegible]

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian, dapat disimpulkan bahwa strategi **Greedy All In Diamond** merupakan pendekatan paling adaptif dan efisien dalam permainan Diamond Collector. Pendekatan ini menggabungkan berbagai sub-strategi greedy seperti Safe Collection, Trap Strategy, High-Value First, Patrol, hingga Time-Aware Strategy, yang memungkinkan bot beradaptasi secara cerdas terhadap perubahan kondisi permainan. Strategi ini terbukti menghasilkan keputusan cepat dan efektif, baik dalam menghindari musuh, memprioritaskan diamond bernilai tinggi, maupun dalam menjaga keamanan skor saat waktu hampir habis. Melalui pengujian yang mencakup berbagai skenario permainan—seperti pengumpulan di area clustering, pemanfaatan portal, hingga penghindaran musuh—bot menunjukkan performa konsisten dan responsif terhadap situasi sekitar. Dengan algoritma yang berbasis greedy heuristics dan pemanfaatan Manhattan Distance serta Breadth-First Search (BFS), bot dapat melakukan navigasi dan pengambilan keputusan yang efisien secara waktu dan nilai.

5.2 Saran

Sebagai mahasiswa yang mengerjakan tugas besar ini, kami menyadari bahwa proses pengembangan bot dalam permainan Diamond Collector bukan hanya soal menerapkan algoritma, tetapi juga melatih cara berpikir logis, terstruktur, dan adaptif terhadap tantangan yang terus berubah. Selama pengerjaan laporan ini, kami menyadari pentingnya memahami konsep dasar algoritma greedy secara menyeluruh, serta bagaimana menghubungkannya dengan kondisi nyata di dalam permainan yang bersifat dinamis dan kompetitif.

LAMPIRAN

A. Repository Github ([link](#))

https://github.com/15-069-ZahwaNatasyaHamzah/TUBES_STIGMARC_GEMBOT-14-

B. Video Penjelasan (link GDrive)

<https://drive.google.com/drive/folders/1Q0IvjVJVvErSKgWW5jI9ROwahC5MQ7AQ?usp=sharing>

DAFTAR PUSTAKA

- [1] Sutrisno, E. (2011). *Algoritma dan Pemrograman*. Yogyakarta: Andi.
- [2] Nugroho, A. (2016). *Algoritma dan Struktur Data*. Yogyakarta: Andi.
- [3] Sembiring, R. (2020). *Kecerdasan Buatan: Teori dan Implementasi*. Medan: Perdana Publishing.