

IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RE
di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera



Oleh: Kelompok Beeskuerem (B²M)

Memory Simanjuntak 123140095

Arsa Salsabila 123140108

Grace Exauditha Nababan 123140115

Dosen Pengampu: Imam Ekowicaksono, S.Si., M.Si.

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

2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I	
DESKRIPSI TUGAS.....	3
1.1 Program permainan Diamonds terdiri atas:.....	3
1.2 Komponen Permainan Diamonds.....	3
1.3 Cara Kerja Permainan Diamonds.....	5
BAB II	
LANDASAN TEORI.....	7
2.1 Dasar Teori.....	7
2.2 Cara Kerja Program.....	9
BAB III	
APLIKASI STRATEGI GREEDY.....	14
3.1 Proses Mapping.....	14
3.2 Eksplorasi Alternatif Solusi Greedy.....	15
3.3 Analisis efficiency dan Efektivitas Solusi Greedy.....	19
3.4 Strategi Greedy yang Dipilih.....	22
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	28
4.1 Implementasi Algoritma Greedy.....	28
4.2 Struktur Data yang Digunakan.....	37
4.3 Pengujian Program.....	38
BAB V	
KESIMPULAN DAN SARAN.....	42
5.1 Kesimpulan.....	42
5.2 Saran.....	43
LAMPIRAN.....	44
DAFTAR PUSTAKA.....	45

BAB I

DESKRIPSI TUGAS

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

1.1 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
 - c) <https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>
- 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
 - d) <https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

1.2 Komponen Permainan Diamonds

- **Diamonds**



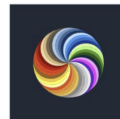
Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahnya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

- **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.

- **Teleporters**



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





- **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.

- **Inventory**

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

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.

1.3 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).

BAB II

LANDASAN TEORI

2.1 Dasar Teori

Dalam dunia komputasi, khususnya dalam penyelesaian masalah optimasi, terdapat berbagai pendekatan algoritmik. Salah satu pendekatan yang populer karena kesederhanaannya dan kecepatan eksekusinya adalah **algoritma greedy**. Greedy secara harfiah berarti “rakus” atau “tamak”, dan memang demikianlah karakteristik pendekatannya: selalu mengambil keputusan terbaik yang tersedia saat ini tanpa mempertimbangkan dampak masa depan.

Secara umum, algoritma greedy bekerja secara **langkah demi langkah (step-by-step)**. Pada setiap langkah, algoritma ini akan memilih **solusi optimum lokal**, yaitu keputusan yang tampak paling menguntungkan saat itu, dengan harapan bahwa kumpulan dari keputusan lokal tersebut akan menghasilkan **solusi optimum global**. Namun demikian, tidak semua persoalan dapat diselesaikan dengan baik menggunakan pendekatan ini, karena greedy tidak menelusuri semua kemungkinan solusi seperti halnya metode brute force atau dynamic programming.

2.1.1 Prinsip Dasar dan Elemen Greedy

Sebuah algoritma greedy memiliki lima komponen utama: (1) *himpunan kandidat* yang mencakup semua kemungkinan pilihan, (2) *himpunan solusi* yang bertahap dibentuk selama proses, (3) *fungsi seleksi* yang menentukan kandidat terbaik yang akan dipilih, (4) *fungsi kelayakan* yang mengecek apakah kandidat dapat dimasukkan ke dalam solusi, dan (5) *fungsi obyektif* yang menjadi ukuran optimalitas, seperti minimisasi waktu atau maksimisasi keuntungan [1].

Skema dasar greedy dimulai dari solusi kosong. Selama solusi belum lengkap dan masih ada kandidat tersisa, algoritma memilih kandidat terbaik menurut strategi seleksi tertentu, mengecek kelayakannya, dan jika layak, menambahkannya ke solusi. Proses ini diulang sampai tidak ada lagi kandidat atau solusi sudah memenuhi kriteria.

2.1.2 Penerapan dan Contoh Kasus

Salah satu contoh klasik penggunaan algoritma greedy adalah dalam **persoalan penukaran uang (coin exchange problem)**. Dalam kasus ini, kita ingin menukar sejumlah nilai uang dengan jumlah minimum koin dari pecahan tertentu. Strategi greedy yang diterapkan adalah dengan **selalu memilih koin dengan nilai terbesar** yang masih bisa dipakai. Untuk sistem koin seperti Rupiah, Dollar, atau Euro, pendekatan ini biasanya memberikan hasil optimal. Namun, untuk sistem koin buatan yang tidak teratur, algoritma greedy bisa gagal memberikan solusi terbaik karena bisa terjebak pada keputusan lokal yang tidak menghasilkan hasil global terbaik [1].

Dalam kasus **knapsack problem**, algoritma greedy dapat digunakan dalam dua varian. Untuk **integer knapsack** (objek tidak dapat dipecah), greedy hanya memberikan solusi hampiran, bukan solusi optimal. Namun, untuk **fractional knapsack** (objek bisa diambil sebagian), greedy **selalu menghasilkan solusi optimal** jika strategi seleksinya berdasarkan **rasio profit per unit bobot (p_i/w_i)**. Hal ini bahkan telah dibuktikan secara matematis oleh Ellis Horowitz dan Sartaj Sahni dalam literatur klasik algoritma [2].

Greedy juga diterapkan dalam **job scheduling dengan tenggat waktu**, yaitu persoalan di mana pekerjaan harus diselesaikan sebelum deadline untuk mendapatkan keuntungan. Strategi greedy-nya adalah **memilih pekerjaan dengan profit terbesar terlebih dahulu**, dan menempatkannya dalam jadwal selama masih memenuhi tenggat. Meskipun tidak selalu optimal, pendekatan ini sering kali menghasilkan hasil mendekati optimal dengan kompleksitas jauh lebih rendah dibanding metode exhaustive search [1].

2.1.3 Kelebihan dan Keterbatasan

Kekuatan utama algoritma greedy terletak pada efficiency waktu dan kesederhanaannya. Pendekatan ini sangat cocok digunakan dalam kasus-kasus besar atau real-time di mana solusi eksak terlalu mahal secara komputasi. Namun, greedy memiliki kelemahan: **ia tidak menjamin solusi optimal**. Keputusan lokal terbaik belum tentu menghasilkan solusi global terbaik, dan dalam beberapa kasus, greedy dapat menghasilkan solusi yang jauh dari optimal.

Oleh karena itu, sebelum menggunakan algoritma greedy, kita harus mengevaluasi terlebih dahulu apakah masalah yang dihadapi memiliki struktur yang memungkinkan greedy memberikan solusi optimal. Dalam kasus tertentu, kita perlu membuktikan secara matematis bahwa pendekatan greedy yang digunakan memang menjamin keoptimalan.

2.2 Cara Kerja Program

Cara kerja program ini adalah dengan merancang sebuah bot yang dapat bergerak secara otomatis dalam papan permainan Diamonds untuk mengumpulkan diamond sebanyak mungkin dengan cara yang efisien. Bot mengambil keputusan menggunakan pendekatan algoritma greedy, yaitu memilih target diamond berdasarkan strategi tertentu, seperti mengejar diamonds yang paling dekat, diamonds yang memiliki nilai paling tinggi, atau menggabungkan antara nilai, jarak, dan faktor keamanan.

Bot memantau kondisi permainan secara langsung melalui informasi yang tersedia, seperti posisi bot, lokasi diamond, keberadaan musuh, serta waktu permainan yang tersisa. Berdasarkan data tersebut, bot menentukan langkah yang akan diambil sesuai dengan strategi yang telah ditentukan. Arah gerakan bot dinyatakan dalam bentuk pasangan nilai (Δx , Δy) yang menunjukkan perubahan posisi menuju target yang dipilih.

2.2.1 Cara Bot Melakukan Aksi

Setiap kali waktu permainan bergerak (setiap tick), bot secara otomatis akan dipanggil untuk menjalankan fungsi `next_move`. Di dalam fungsi ini, bot akan mengakses berbagai informasi penting dari permainan, seperti:

- Posisi bot sendiri dan posisi musuh di papan.
- Lokasi serta nilai dari setiap diamond.
- Jumlah diamond yang telah dikumpulkan oleh bot.
- Sisa waktu yang tersedia dalam permainan.

Setelah memperoleh data tersebut, bot akan menentukan diamond yang paling layak untuk dikejar sesuai dengan strategi yang digunakan. Selanjutnya, bot akan bergerak satu langkah menuju arah diamond yang telah dipilih sebagai target.

2.2.2 Menjalankan Bot Program

- Cara Menjalankan Game Engine

- a. Requirement yang harus di-install

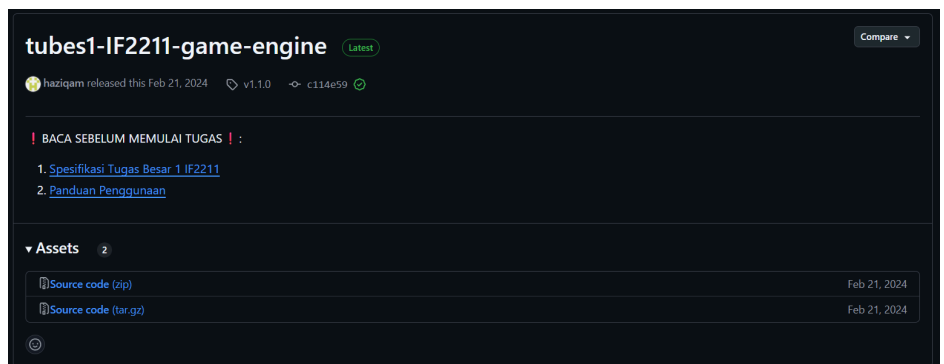
- Node.js ([Node.js — Run JavaScript Everywhere](#))
- Docker
- Yarn

desktop

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

- b. Instalasi dan konfigurasi awal

- 1) Download source code (.zip) pada <https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>



- 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. Build

```
npm run build
```

D. Run

```
npm run start
```

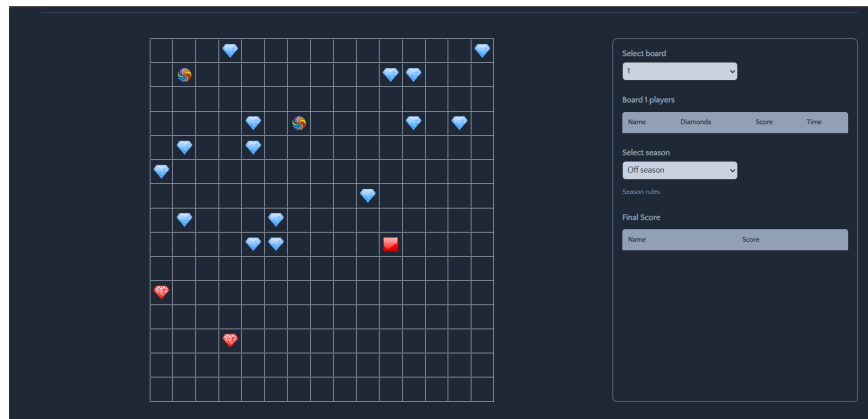
Jika berhasil, tampilan terminal akan terlihat seperti gambar di bawah ini.

```

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): dist\**\*.env
[nodemon] watching extensions: ts,map,js,json
[nodemon] starting `nest start`
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [NestFactory] Starting Nest application...
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [InstanceLoader] AppModule dependencies initialized +25ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RoutesResolver] BoardsController (/api/boards): +101ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/boards, GET} route +3ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/boards/:id, GET} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RoutesResolver] BotsController (/api/bots): +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/bots/:id, GET} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/bots, POST} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/bots/recover, POST} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/bots/:id/join, POST} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/bots/:id/move, POST} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RoutesResolver] HighscoresController (/api/highscores): +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/highscores/:seasonId, GET} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RoutesResolver] RecordingsController (/api/recordings): +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/recordings/seasons/:seasonId, GET} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/recordings/score/last, GET} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/recordings/:id, GET} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RoutesResolver] SeasonsController (/api/seasons): +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/seasons, GET} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/seasons/current, GET} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/seasons/:id, GET} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/seasons/:id/rules, GET} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RoutesResolver] SlackController (/api/slack): +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/slack/seasons, POST} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/slack/season, POST} route +1ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/slack/teams, POST} route +0ms
[Nest] 27132 - 01/06/2025, 14.16.35 LOG [RouterExplorer] Mapped {/api/slack/team, POST} route +0ms

```

Kunjungi frontend melalui <http://localhost:8082/> Berikut adalah tampilan awal frontend.



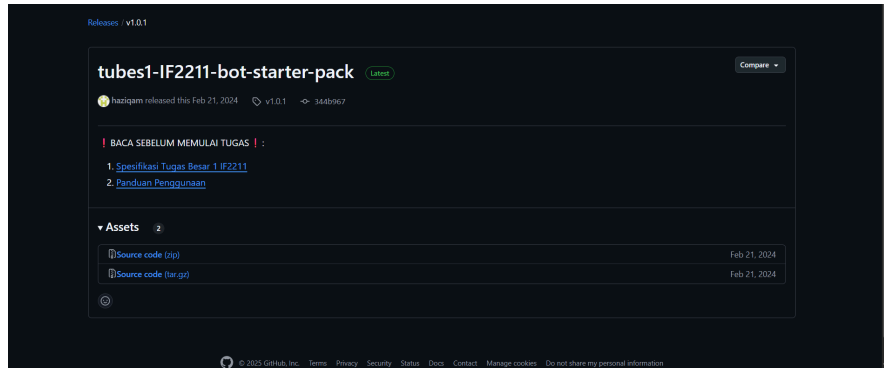
- **Cara Menjalankan Bot**

- a. Requirement yang harus di-install

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

- b. Instalasi dan konfigurasi awal

- 1) Download source code (.zip) pada <https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>



- 2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal
- 3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

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

- 4) Install dependencies menggunakan pip

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

c. Run

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

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

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
```

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Proses mapping dalam aplikasi strategi greedy untuk bot diamond berfungsi sebagai jembatan penghubung antara data mentah dari game engine dengan keputusan strategis yang harus dibuat oleh bot. Ketika game engine mengirimkan informasi berupa objek `'board_bot'` dan `'board'`, bot melakukan tahap pertama mapping dengan mengekstrak data penting seperti posisi saat ini (`'current_position'`) dan properti bot (`'props'`). Data-data ini kemudian dipetakan ke dalam struktur internal bot yang lebih mudah diproses untuk pengambilan keputusan.

Tahap kedua mapping melibatkan kategorisasi objek-objek di lingkungan game menjadi elemen-elemen strategis yang relevan. Bot menganalisis seluruh `'board.game_objects'` dan memetakannya menjadi dua kategori utama: `'enemies'` (bot musuh yang berpotensi berbahaya) dan `'diamonds'` (target yang harus dikumpulkan). Proses filtering ini penting karena mengubah data kompleks menjadi informasi yang dapat langsung digunakan untuk strategi greedy, di mana bot hanya fokus pada objek-objek yang mempengaruhi keputusan berikutnya.

Mapping prioritas merupakan aspek paling krusial dalam implementasi strategi greedy ini. Bot memetakan situasi game saat ini ke dalam hierarki keputusan berlapis: keselamatan dari musuh sebagai prioritas tertinggi, manajemen inventori (kembali ke base saat membawa 4+ diamond), pengelolaan posisi (menjauh dari base setelah menyimpan diamond), pengumpulan diamond baru, dan eksplorasi sebagai prioritas terendah. Setiap kondisi game dipetakan ke dalam salah satu kategori prioritas ini, memastikan bot selalu mengambil keputusan yang paling optimal untuk situasi saat ini.

Inti dari strategi greedy terletak pada fungsi `'find_best_diamond()'` yang melakukan mapping kompleks dari multiple factors menjadi single score. Setiap diamond dipetakan berdasarkan jarak Manhattan, nilai poin, dan tingkat risiko dari musuh terdekat. Formula greedy `'score = distance / diamond_points'` kemudian dimodifikasi dengan faktor keamanan (diamond tidak aman mendapat penalti 2x lipat) dan faktor nilai (diamond bernilai tinggi mendapat bonus 0.7x). Mapping ini memungkinkan bot memilih diamond dengan score terendah, yang berarti diamond dengan kombinasi terbaik antara jarak dekat, nilai tinggi, dan keamanan optimal.

Tahap akhir mapping mengonversi keputusan strategis kompleks menjadi command sederhana yang dapat dipahami game engine. Setelah bot menentukan target terbaik melalui proses greedy, mapping dilakukan untuk mengubah koordinat target menjadi tuple gerakan `(delta_x, delta_y)` menggunakan fungsi `get_direction()`. Proses ini memastikan bahwa strategi canggih yang melibatkan analisis risiko, optimasi nilai, dan perencanaan jangka pendek dapat diterjemahkan menjadi instruksi gerakan yang konkret dan dapat dieksekusi oleh game engine.

3.2 Eksplorasi Alternatif Solusi Greedy

1. Greedy By Efficiency

Kode ini mengimplementasikan bot untuk game pengumpulan berlian menggunakan algoritma greedy yang berfokus pada efficiency. Bot memiliki sistem prioritas berlapis: pertama menghindari musuh dalam radius bahaya, kemudian mengembalikan berlian ke base jika sudah mengumpulkan 4 atau lebih, lalu mencari dan mengumpulkan berlian dengan sistem scoring yang mempertimbangkan jarak dan nilai berlian. Bot juga memiliki mekanisme eksplorasi cerdas yang mengarah ke pusat papan permainan daripada bergerak acak. Keseluruhan strategi dirancang untuk memaksimalkan efficiency pengumpulan berlian sambil meminimalkan risiko dari musuh.

Pemetaan Elemen Greedy

Himpunan Kandidat: Semua berlian yang tersedia di papan permainan (diamonds dari fungsi `get_diamonds()`)

Himpunan Solusi: Berlian yang dipilih untuk dikumpulkan berdasarkan algoritma greedy (`best_diamond` yang dipilih)

Fungsi Solusi: Memilih berlian terbaik berdasarkan score terendah dari kombinasi jarak dan nilai berlian

Fungsi Seleksi: `find_best_diamond()` yang menghitung score untuk setiap berlian dan memilih yang terbaik

Fungsi Kelayakan: Pengecekan keamanan berlian dari musuh (`diamond_safe`) dan validasi properti berlian

Fungsi Objektif: Minimisasi score yang dihitung dengan formula $\text{distance} / \text{diamond_points}$ dengan penalti untuk berlian tidak aman dan bonus untuk berlian bernilai tinggi

2. Greedy By Safe

Kode ini mengimplementasikan bot berlian dengan algoritma greedy yang memprioritaskan keselamatan dalam pengambilan keputusan. Bot menggunakan formula unik $(\text{value} \times \text{safety}) / \text{distance}$ untuk mengevaluasi berlian terbaik, dimana faktor safety dihitung berdasarkan jarak minimum ke musuh dengan formula $\min(1.0, \max(0.1, \text{min_enemy_distance} / 10.0))$. Sistem prioritas bot lebih sederhana: kembali ke base saat membawa 5 berlian, mencari berlian dengan score terbaik, atau kembali ke base jika ada berlian. Bot juga memiliki mekanisme pergerakan aman yang memilih arah dengan tingkat keselamatan tertinggi saat eksplorasi.

Pemetaan Elemen Greedy

Himpunan Kandidat: Semua berlian yang tersedia di papan permainan (diamonds dari fungsi `get_all_diamonds()`)

Himpunan Solusi: Berlian yang dipilih berdasarkan score tertinggi dari formula greedy (`best_diamond`)

Fungsi Solusi: Memilih berlian dengan score tertinggi menggunakan formula $(\text{value} \times \text{safety}) / \text{distance}$

Fungsi Seleksi: `find_best_safe_diamond()` yang menghitung score untuk setiap berlian dan memilih yang tertinggi

Fungsi Kelayakan: Validasi berlian yang dapat diakses dan perhitungan faktor keselamatan

Fungsi Objektif: Maksimisasi score dengan formula $(\text{value} \times \text{safety}) / \text{distance}$ yang mengoptimalkan nilai, keselamatan, dan kedekatan

3. Greedy By Ratio

Kode ini mengimplementasikan bot berlian dengan algoritma greedy yang fokus pada efficiency maksimal melalui optimisasi rasio $\text{value}/\text{distance}$. Bot memiliki

strategi sederhana namun efektif: kembali ke base saat membawa 5 berlian, mencari berlian dengan efficiency tertinggi menggunakan formula $\text{value}/(\text{distance}+1)$, kemudian kembali ke base jika masih membawa berlian. Algoritma ini murni greedy tanpa pertimbangan keselamatan atau faktor eksternal lainnya, sehingga fokus sepenuhnya pada maksimisasi return per unit jarak yang ditempuh. Bot menggunakan eksplorasi acak ketika tidak ada objective yang jelas.

Pemetaan Elemen Greedy

Himpunan Kandidat: Semua berlian yang tersedia di papan permainan (diamonds dari fungsi `get_diamonds()`)

Himpunan Solusi: Berlian yang dipilih berdasarkan efficiency tertinggi (`best_diamond`)

Fungsi Solusi: Memilih berlian dengan efficiency maksimal menggunakan formula $\text{value}/(\text{distance}+1)$

Fungsi Seleksi: `find_most_efficient_diamond()` yang menghitung efficiency setiap berlian dan memilih yang tertinggi

Fungsi Kelayakan: Validasi berlian yang dapat diakses dan memiliki nilai valid

Fungsi Objektif: Maksimisasi efficiency dengan formula $\text{value}/(\text{distance}+1)$ untuk mengoptimalkan return per unit effort

4. Greedy By Highest Value

Kode ini mengimplementasikan bot berlian dengan algoritma greedy yang paling sederhana namun fundamental: selalu mengejar berlian dengan nilai tertinggi. Bot memiliki strategi linear yang fokus pada maksimisasi value absolut tanpa mempertimbangkan jarak atau faktor lainnya. Sistem prioritas bot sangat straightforward: kembali ke base saat membawa 5 berlian, mencari berlian bernilai tertinggi, kemudian kembali ke base jika masih ada berlian. Algoritma ini menggunakan strategi "value-first" yang mengabaikan efficiency perjalanan dan hanya fokus pada accumulation berlian premium (red diamonds dengan nilai 2 poin).

Pemetaan Elemen Greedy

Himpunan Kandidat: Semua berlian yang tersedia di papan permainan (diamonds dari fungsi `get_diamonds()`)

Himpunan Solusi: Berlian dengan nilai tertinggi yang dipilih (`best_diamond`)

Fungsi Solusi: Memilih berlian dengan nilai maksimal tanpa pertimbangan lain

Fungsi Seleksi: `find_highest_value_diamond()` yang mencari berlian dengan `max_value`

Fungsi Kelayakan: Validasi berlian yang dapat diakses dan memiliki nilai valid

Fungsi Objektif: Maksimisasi nilai absolut berlian dengan prioritas pada high-value diamonds

5. Greedy By Nearest

Kode ini mengimplementasikan bot berlian dengan algoritma greedy paling dasar: selalu mengejar berlian terdekat tanpa mempertimbangkan nilai. Bot menggunakan strategi "nearest-first" yang memprioritaskan minimisasi jarak tempuh untuk mencapai efficiency movement maksimal. Sistem prioritas bot sangat sederhana: kembali ke base saat membawa 5 berlian, mencari berlian terdekat menggunakan Manhattan distance, kemudian kembali ke base jika masih ada berlian. Algoritma ini menggunakan prinsip greedy murni pada aspek spasial, mengabaikan sepenuhnya value optimization dan hanya fokus pada proximity-based decision making.

Pemetaan Elemen Greedy GreedyNearestBot

Himpunan Kandidat: Semua berlian yang tersedia di papan permainan (diamonds dari fungsi `get_diamonds()`)

Himpunan Solusi: Berlian dengan jarak terdekat yang dipilih (`nearest_diamond`)

Fungsi Solusi: Memilih berlian dengan jarak minimal dari posisi current

Fungsi Seleksi: `find_nearest_diamond()` yang mencari berlian dengan `min_distance`

Fungsi Kelayakan: Validasi berlian yang dapat diakses dan memiliki posisi valid

Fungsi Objektif: Minimisasi jarak tempuh menggunakan Manhattan distance untuk efficiency movement

3.3 Analisis efficiency dan Efektivitas Solusi Greedy

a. Greedy by efficiency

1. Analisis efficiency Solusi

Kompleksitas waktu algoritma adalah $O(n \times m)$ dimana n adalah jumlah berlian dan m adalah jumlah musuh, karena untuk setiap berlian perlu dicek keamanannya terhadap semua musuh. Fungsi `find_best_diamond()` memiliki kompleksitas $O(n \times m)$ dan fungsi `get_enemies()` memiliki kompleksitas $O(k)$ dimana k adalah total objek di papan. Secara keseluruhan, algoritma cukup efisien untuk ukuran papan permainan yang wajar karena operasi dilakukan dalam satu pass tanpa rekursi atau nested loop yang kompleks.

2. Analisis Efektivitas Solusi

Solusi ini cukup efektif karena menggabungkan strategi greedy dengan pertimbangan taktis. Sistem prioritas yang terstruktur memastikan bot dapat bertahan hidup (menghindari musuh) sambil tetap produktif mengumpulkan berlian. Penggunaan threshold 4 berlian untuk kembali ke base mencegah kehilangan terlalu banyak berlian jika terkena musuh. Fungsi scoring yang mempertimbangkan rasio jarak terhadap nilai berlian memberikan keputusan yang rasional. Namun, kelemahan potensial terletak pada sifat greedy yang mungkin terjebak dalam optimum lokal dan tidak mempertimbangkan strategi jangka panjang seperti pola pergerakan musuh atau koordinasi dengan bot lain.

b. greedy by safe

1. Analisis efficiency Solusi

Kompleksitas waktu algoritma adalah $O(n \times m)$ dimana n adalah jumlah berlian dan m adalah jumlah musuh, karena untuk setiap berlian perlu dihitung faktor keselamatan terhadap semua musuh. Fungsi `find_best_safe_diamond()` memiliki kompleksitas $O(n \times m)$ dan fungsi `safe_move()` memiliki kompleksitas $O(d \times m)$ dimana d adalah jumlah arah gerakan (konstanta 4). Secara keseluruhan, algoritma cukup efisien karena

menggunakan pendekatan single-pass tanpa nested loop kompleks. Perhitungan safety factor yang terintegrasi menghindari redundansi operasi dibandingkan dengan sistem penghindaran musuh terpisah.

2. Analisis Efektivitas Solusi

Solusi ini sangat efektif untuk lingkungan permainan dengan tingkat ancaman tinggi karena faktor keselamatan terintegrasi dalam setiap keputusan strategis. Formula $(\text{value} \times \text{safety}) / \text{distance}$ memberikan keseimbangan optimal antara reward (nilai berlian), risk (keselamatan), dan effort (jarak tempuh). Threshold 5 berlian untuk kembali ke base memberikan keamanan ekstra namun mungkin kurang agresif dibanding bot lain. Kelemahan utama adalah tidak ada mekanisme penghindaran musuh reaktif, sehingga bot mungkin kesulitan dalam situasi chase langsung. Namun, pendekatan proaktif dengan mempertimbangkan keselamatan di setiap langkah membuat bot ini konsisten dan dapat diandalkan dalam jangka panjang.

c. greedy by ratio

1. Analisis efficiency Solusi

Kompleksitas waktu algoritma adalah $O(n)$ dimana n adalah jumlah berlian, karena setiap berlian dievaluasi sekali tanpa dependensi pada objek lain. Fungsi `find_most_efficient_diamond()` memiliki kompleksitas linear $O(n)$ dan tidak ada nested loop atau operasi kompleks. Algoritma ini paling efisien di antara ketiga bot karena tidak melakukan perhitungan safety factor atau pengecekan musuh. Overhead komputasi minimal karena hanya melakukan perhitungan matematika sederhana (pembagian) untuk setiap berlian. Struktur kode yang simpel juga mengurangi risiko error dan meningkatkan maintainability.

2. Analisis Efektivitas Solusi

Solusi ini sangat efektif dalam lingkungan dengan risiko rendah karena fokus murni pada optimisasi ekonomi tanpa gangguan faktor eksternal. Formula $\text{value}/\text{distance}$ memberikan keputusan yang rasional secara matematis untuk memaksimalkan profit per movement. Threshold 5 berlian memberikan keseimbangan yang baik antara safety dan productivity. Namun, kelemahan utama adalah tidak ada pertimbangan keselamatan sama sekali, sehingga bot rentan terhadap musuh. Algoritma ini cocok untuk lingkungan

kompetitif dengan sedikit ancaman atau dalam fase awal permainan ketika fokus pada accumulation lebih penting daripada survival. Efektivitas optimal tercapai ketika berlian bernilai tinggi tersebar merata dan tidak ada tekanan dari kompetitor.

d. greedy by highest value

1. Analisis efficiency Solusi

Kompleksitas waktu algoritma adalah $O(n)$ dimana n adalah jumlah berlian, karena setiap berlian dievaluasi sekali untuk mencari nilai maksimal. Fungsi `find_highest_value_diamond()` melakukan single-pass linear search dengan kompleksitas $O(n)$. Algoritma ini memiliki efficiency komputasi tertinggi di antara semua bot karena hanya melakukan operasi comparison sederhana tanpa perhitungan matematika kompleks. Tidak ada overhead untuk distance calculation, safety factor, atau enemy detection. Memory usage minimal karena hanya menyimpan reference ke best diamond dan max value saat iterasi.

2. Analisis Efektivitas Solusi

Solusi ini memiliki efektivitas terbatas karena mengabaikan aspek spatial efficiency yang krusial dalam game strategy. Bot akan bergerak ke berlian bernilai tinggi meskipun jaraknya sangat jauh, mengakibatkan inefficient movement dan vulnerability terhadap kompetitor. Keunggulan utama adalah konsistensi dalam prioritizing high-value targets, yang optimal ketika red diamonds (nilai 2) tersebar dengan density rendah. Namun, dalam skenario dimana blue diamonds (nilai 1) lebih accessible dan abundant, bot ini akan suffer karena mengorbankan multiple low-value diamonds untuk satu high-value diamond yang mungkin contested. Efektivitas maksimal tercapai dalam early game atau ketika player dominance sudah established.

e. greedy by nearest

1. Analisis efficiency Solusi

Kompleksitas waktu algoritma adalah $O(n)$ dimana n adalah jumlah berlian, karena setiap berlian dievaluasi sekali untuk menghitung jarak Manhattan. Fungsi `find_nearest_diamond()` melakukan single-pass linear search dengan perhitungan distance yang konstan $O(1)$ per berlian. Algoritma ini memiliki efficiency komputasi tinggi karena hanya melakukan operasi aritmatika sederhana (penjumlahan absolute values) tanpa

complex mathematical operations. Memory footprint minimal karena hanya menyimpan reference ke nearest diamond dan min distance. Overhead sangat rendah karena tidak ada safety calculations, value comparisons, atau external factor considerations.

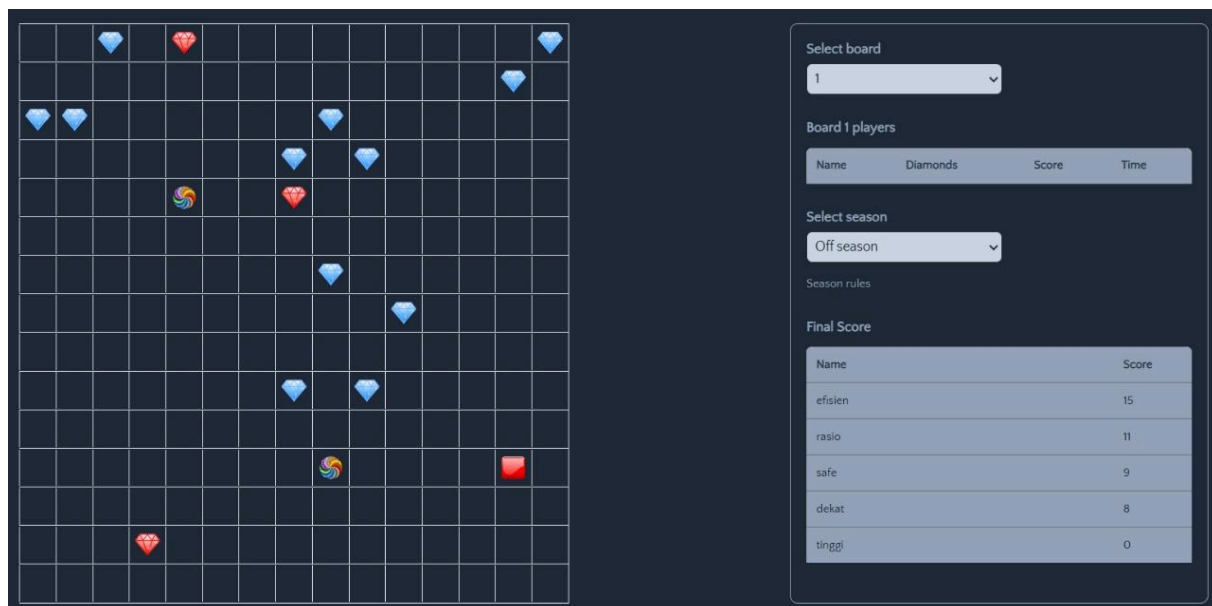
2. Analisis Efektivitas Solusi

Solusi ini memiliki efektivitas yang bervariasi tergantung distribusi berlian dan competitive landscape. Keunggulan utama adalah movement efficiency yang optimal dalam terms of distance traveled, menghasilkan fast collection cycle dan reduced exposure time. Bot ini excellent untuk scenarios dengan uniform diamond distribution dan low competition. Namun, kelemahan signifikan terletak pada value blindness - bot akan mengoleksi multiple low-value diamonds sambil mengabaikan high-value diamonds yang sedikit lebih jauh. Dalam competitive environment, strategi ini vulnerable terhadap value-focused competitors yang dapat mengklaim high-value targets. Efektivitas optimal tercapai ketika diamond density tinggi dan value distribution relatif uniform.

3.4 Strategi Greedy yang Dipilih

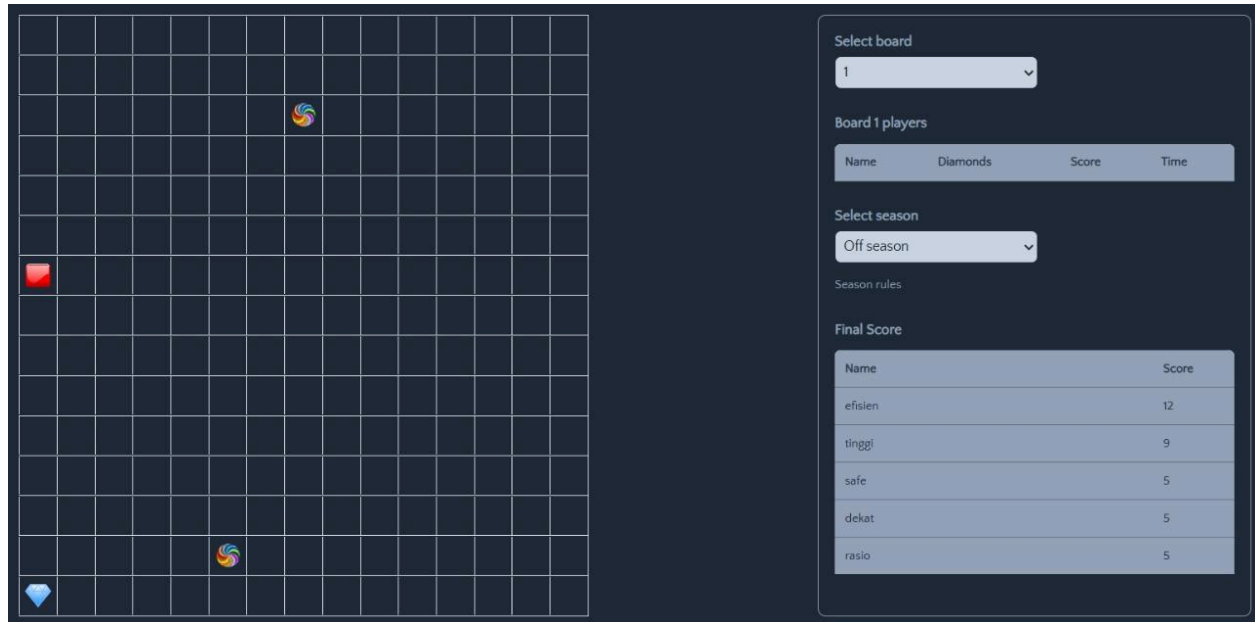
Dari hasil pertandingan yang dilakukan sebanyak lima kali, terlihat bahwa terjadi perbedaan performa yang ditandai dengan perubahan peringkat yang diperoleh tiap alternatif dalam setiap pertandingannya.

Laga 1



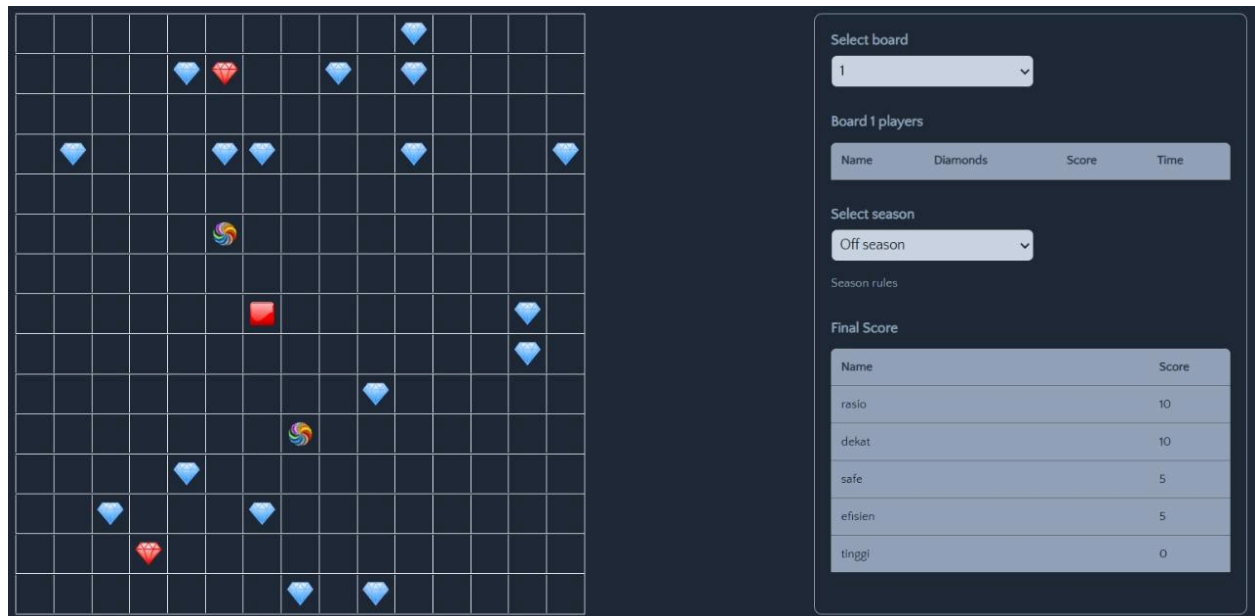
Gambar 3.4.1 Hasil Pertandingan Antar Alternative 1

Laga 2



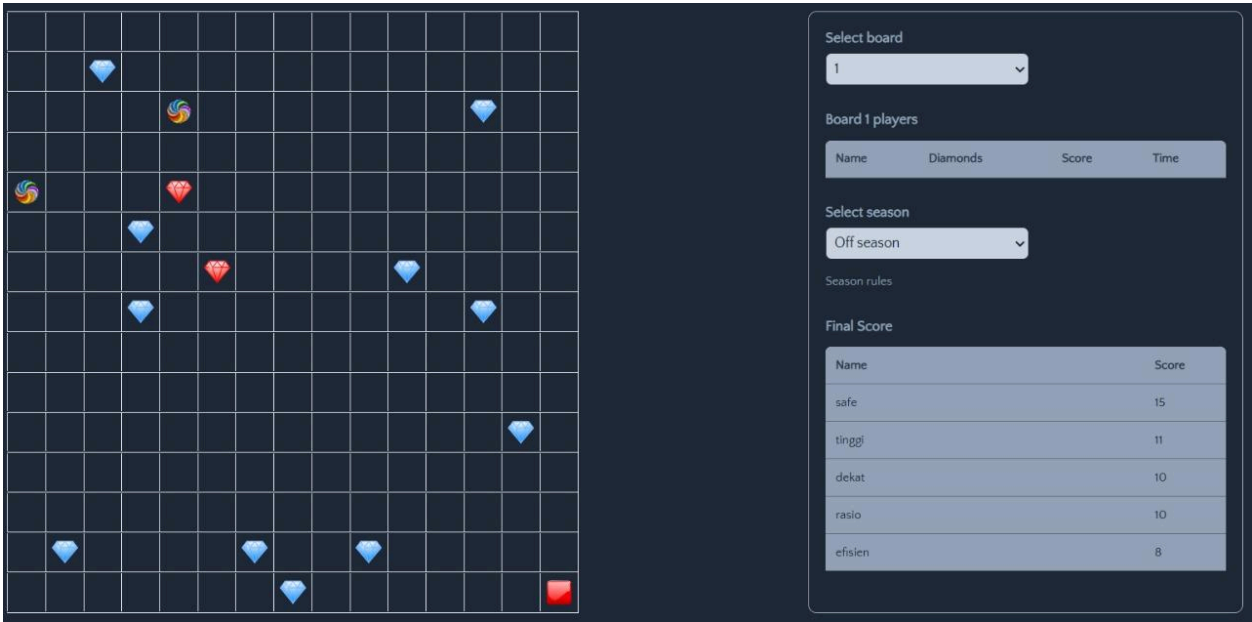
Gambar 3.4.2 Hasil Pertandingan Antar Alternative 2

Laga 3



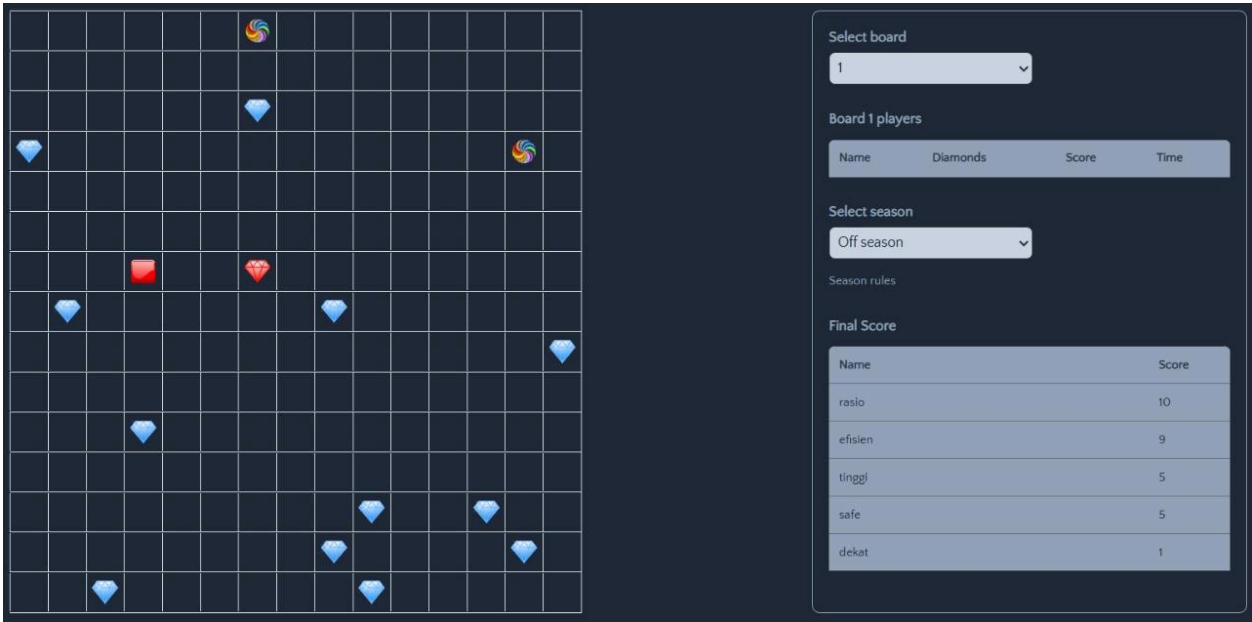
Gambar 3.4.3 Hasil Pertandingan Antar Alternative 3

Laga 4



Gambar 3.4.4 Hasil Pertandingan Antar Alternative 4

Laga 5



Gambar 3.4.5 Hasil Pertandingan Antar Alternative 5

Tabel 3.4.1 Analisis perbandingan bot

Bot	Laga 1	Laga 2	Laga 3	Laga 4	Laga 5	Rata-rata
Greedy by Efficiency	15	12	5	8	9	9.8
Greedy by Ratio	11	5	10	10	10	9.2
Greedy by Safe	9	5	5	14	5	7.6
Greedy by Nearest	8	5	10	10	1	6.8
Greedy by Highest Value	0	9	0	11	5	5.0

Berdasarkan berbagai pendekatan greedy yang telah diuraikan dapat disimpulkan bahwa masing-masing strategi memiliki keunggulan dan kelemahan tersendiri. Setelah melalui pertimbangan dan pengujian, strategi yang dipilih untuk diimplementasikan adalah **Greedy by Efficiency** dengan pendekatan:

- Enemy detection dengan radius 3, *safe distance* 2
- Return to base pada 4 diamond (bukan 5)
- Kombinasi distance, value, safety dengan weighted penalties/bonuses
- Exploration menuju board center, bukan random
- Behavioral Adaptation: Berbeda-beda response berdasarkan game state

Inilah bot yang paling kompleks dan paling cerdas. Ia mempertimbangkan **jarak, nilai diamond, dan ancaman musuh** sekaligus. Ia juga punya logika untuk kembali ke markas sebelum terlambat dan bisa mengeksplorasi peta dengan arah yang masuk akal. Strategi ini seperti pemain

profesional yang tahu kapan harus menyerang, kapan harus mundur, dan kapan harus menghindar. Ia tidak hanya bertindak reaktif, tapi juga punya rencana jangka panjang. Dari semua percobaan, bot ini memang mencetak skor tertinggi, walaupun kadang performanya naik-turun. Tapi secara umum, pendekatan adaptif dan multi-faktornya membuatnya unggul dalam **lingkungan permainan yang kompetitif dan dinamis**.

Beberapa strategi lain dipertimbangkan namun tidak digunakan. Misalnya:

1. **Greedy by Ratio**, strategi ini berusaha mencari keseimbangan terbaik antara nilai diamond dan jaraknya. Bot menghitung rasio antara poin yang akan diperoleh dan langkah yang harus ditempuh. Pendekatan ini seperti memilih makanan yang paling bergizi tapi juga paling mudah diambil. Keunggulan dari strategi ini adalah konsistensi. Bot tidak terlalu ambisius, tapi juga tidak bodoh. Ia bisa beradaptasi di banyak situasi dan menunjukkan hasil yang stabil dalam berbagai kondisi. Namun, karena tidak mempertimbangkan keberadaan musuh, bot ini tetap rentan diserang dalam situasi kompetitif.
2. **Greedy by Safe**, bot ini sangat berhati-hati. Ia akan mengecek dulu apakah suatu diamond berada di area yang aman dari musuh. Ia baru akan mendekat jika merasa tempat itu cukup aman. Pendekatan ini seperti orang yang hanya mau pergi ke toko kalau tahu jalannya sepi dan aman. Hasilnya adalah bot yang sangat defensif. Dalam beberapa kasus, ia bisa mencetak skor tinggi karena berhasil bertahan dan menghindari konflik. Tapi kadang juga terlalu hati-hati, sampai melewatkan peluang bagus. Strategi ini cocok untuk pemain yang ingin main aman dan hindari konfrontasi.
3. **Greedy by Nearest**, strategi ini mengambil pendekatan yang sangat sederhana. Bot akan selalu menuju diamond yang paling dekat, tanpa melihat nilai atau kondisi sekitar. Dalam praktiknya, strategi ini seperti seseorang yang memilih makanan paling dekat di meja tanpa melihat apakah itu makanan enak atau tidak. Keuntungannya jelas: cepat, ringan, dan mudah diprogram. Tapi, pendekatan ini kurang pintar. Bot bisa terjebak berputar-putar di area diamond murah atau malah mengambil risiko besar karena tidak tahu ada musuh di dekatnya. Dalam kompetisi yang ketat, strategi ini cenderung kalah karena tidak mempertimbangkan konteks permainan.

4. **Greedy by Highest Value**, bot ini punya satu tujuan yaitu cari diamond dengan poin tertinggi, biasanya diamond merah. Strategi ini seperti seseorang yang hanya mau ambil barang paling mahal di toko, walaupun tempatnya jauh atau susah dijangkau. Hasilnya bisa sangat menguntungkan, tapi juga sangat berisiko. Jika diamond mahalnya jauh atau dijaga musuh, bot ini bisa gagal total. Dari hasil percobaan, terlihat bahwa bot ini punya **volatilitas paling tinggi**, kadang dapat nilai besar, kadang tidak dapat apa-apa. Ini adalah strategi yang cocok untuk pemain yang suka ambil risiko tinggi dengan imbal hasil tinggi.

Meskipun terdapat berbagai alternatif strategi yang telah diuji, bot yang menggunakan pendekatan **Greedy by Efficiency** terbukti memberikan kinerja yang paling stabil dan adaptif dalam berbagai situasi permainan. Hal ini terlihat dari kemampuannya untuk secara konsisten meraih skor tinggi, bahkan ketika tata letak papan dan posisi musuh berubah-ubah. Dari hasil beberapa percobaan, bot dengan strategi ini lebih sering menempati peringkat atas dibandingkan bot dengan strategi lainnya, serta menunjukkan ketahanan yang baik saat menghadapi tekanan dalam permainan.

Dengan memperhitungkan berbagai faktor seperti konsistensi performa, efektivitas pergerakan, dan perlindungan terhadap ancaman musuh, dapat disimpulkan bahwa **greedy by efficiency** merupakan strategi yang paling tepat untuk diimplementasikan dalam tugas ini. Pendekatan ini terbukti mampu menghasilkan performa yang unggul secara konsisten dan tetap relevan terhadap dinamika permainan yang terus berubah.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

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

class GreedyByEfficiency(BaseLogic):
    def __init__(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] = None
        self.target_diamond: Optional[GameObject] = None
        self.danger_radius = 3 # Radius untuk mendeteksi musuh
        self.safe_distance = 2 # Jarak aman minimum dari musuh

    def next_move(self, board_bot: GameObject, board: Board):
        try:
            props = board_bot.properties
            current_position = board_bot.position

            # Prioritas 1: Menghindari musuh - periksa ancaman terdekat
            enemies = self.get_enemies(board, board_bot)
            if enemies:
                danger_move = self.avoid_enemies(current_position, enemies)
                if danger_move:
                    return danger_move

            # Prioritas 2: Jika membawa 4+ berlian, kembali ke base (dikurangi dari 5)
            if props.diamonds >= 4:
                base = props.base
                if base and not self.is_at_position(current_position, base):
                    delta_x, delta_y = get_direction(
```

```

        current_position.x,
        current_position.y,
        base.x,
        base.y,
    )
    return delta_x, delta_y

# Prioritas 3: Jika di base dan punya berlian, bergerak menjauh dulu
if props.base and self.is_at_position(current_position, props.base) and props.diamonds > 0:
    # Bergerak menjauh dari base untuk melanjutkan pengumpulan
    return self.move_away_from_base(current_position, props.base)

# Prioritas 4: Cari dan kumpulkan berlian
diamonds = self.get_diamonds(board)

if diamonds:
    # Cari berlian terbaik dengan penilaian yang diperbaiki
    best_diamond = self.find_best_diamond(current_position, diamonds, enemies)
    if best_diamond:
        delta_x, delta_y = get_direction(
            current_position.x,
            current_position.y,
            best_diamond.position.x,
            best_diamond.position.y,
        )
        return delta_x, delta_y

# Prioritas 5: Jika membawa berlian dan tidak ada berlian bagus yang ditemukan, kembali ke base
if props.diamonds > 0:
    base = props.base
    if base and not self.is_at_position(current_position, base):
        delta_x, delta_y = get_direction(
            current_position.x,
            current_position.y,
            base.x,
            base.y,
        )
        return delta_x, delta_y

```

```

# Prioritas 6: Eksplorasi cerdas alih-alih gerakan acak
return self.get_exploration_move(current_position, board)

except Exception as e:
    # Fallback ke gerakan acak yang aman
    print(f"Error in GreedyDiamondBot: {e}")
    return self.get_safe_random_move()

def get_enemies(self, board: Board, bot: GameObject) -> List[GameObject]:
    """Dapatkan semua bot musuh di papan"""
    enemies = []
    try:
        for game_object in board.game_objects:
            # Periksa apakah itu bot dan bukan bot kita
            if (hasattr(game_object, 'type') and 'Bot' in game_object.type and
                game_object.id != bot.id):
                # Periksa apakah musuh dalam radius bahaya
                distance = self.manhattan_distance(bot.position, game_object.position)
                if distance <= self.danger_radius:
                    enemies.append(game_object)
    except:
        pass
    return enemies

def avoid_enemies(self, current_pos: Position, enemies: List[GameObject]) -> Optional[Tuple[int, int]]:
    """Hitung gerakan untuk menghindari musuh"""
    if not enemies:
        return None

    # Cari musuh terdekat
    closest_enemy = min(enemies,
                        key=lambda enemy: self.manhattan_distance(current_pos, enemy.position))

    enemy_pos = closest_enemy.position
    distance = self.manhattan_distance(current_pos, enemy_pos)

    # Jika terlalu dekat, bergerak menjauh segera

```

```

if distance <= self.safe_distance:
    # Hitung arah menjauhi musuh
    if enemy_pos.x != current_pos.x:
        move_x = -1 if enemy_pos.x > current_pos.x else 1
    else:
        move_x = 0

    if enemy_pos.y != current_pos.y:
        move_y = -1 if enemy_pos.y > current_pos.y else 1
    else:
        move_y = 0

    # Prioritaskan sumbu dengan selisih jarak yang lebih besar
    if abs(enemy_pos.x - current_pos.x) > abs(enemy_pos.y - current_pos.y):
        return (move_x, 0)
    else:
        return (0, move_y)

```

```

return None

```

```

def get_diamonds(self, board: Board) -> List[GameObject]:
    """Dapatkan semua berlian di papan"""
    diamonds = []
    try:
        for game_object in board.game_objects:
            if hasattr(game_object, 'type') and game_object.type == "DiamondGameObject":
                diamonds.append(game_object)
            elif "Diamond" in str(type(game_object)):
                diamonds.append(game_object)
    except:
        pass
    return diamonds

```

```

def find_best_diamond(self, current_pos: Position, diamonds: List[GameObject],
    enemies: List[GameObject]) -> Optional[GameObject]:
    """Cari berlian terbaik menggunakan algoritma greedy yang diperbaiki"""
    if not diamonds:
        return None

```

```

best_diamond = None
best_score = float('inf')

for diamond in diamonds:
    try:
        # Hitung jarak Manhattan
        distance = self.manhattan_distance(current_pos, diamond.position)

        # Dapatkan poin berlian jika tersedia
        diamond_points = 1
        if hasattr(diamond, 'properties') and hasattr(diamond.properties, 'points'):
            diamond_points = diamond.properties.points

        # Periksa apakah berlian aman dari musuh
        diamond_safe = True
        for enemy in enemies:
            enemy_to_diamond_dist = self.manhattan_distance(enemy.position, diamond.position)
            if enemy_to_diamond_dist <= self.safe_distance:
                diamond_safe = False
                break

        # Hitung skor: semakin rendah semakin baik
        if diamond_points > 0:
            score = distance / diamond_points
        else:
            score = distance

        # Penalti untuk berlian yang tidak aman
        if not diamond_safe:
            score *= 2 # Buat berlian tidak aman kurang menarik

        # Bonus untuk berlian bernilai tinggi
        if diamond_points >= 2:
            score *= 0.7 # Buat berlian bernilai tinggi lebih menarik

        if score < best_score:
            best_score = score

```



```

        best_diamond = diamond

    except Exception:
        continue

    return best_diamond

def is_at_position(self, pos1: Position, pos2: Position) -> bool:
    """Periksa apakah dua posisi sama"""
    return pos1.x == pos2.x and pos1.y == pos2.y

def move_away_from_base(self, current_pos: Position, base: Position) -> Tuple[int, int]:
    """Bergerak menjauh dari base untuk melanjutkan eksplorasi"""
    # Hitung arah menjauhi base
    if base.x != current_pos.x:
        move_x = -1 if base.x > current_pos.x else 1
    else:
        move_x = random.choice([-1, 1])

    if base.y != current_pos.y:
        move_y = -1 if base.y > current_pos.y else 1
    else:
        move_y = random.choice([-1, 1])

    # Pilih arah yang paling jauh dari base
    if abs(move_x) > abs(move_y):
        return (move_x, 0)
    else:
        return (0, move_y)

def get_exploration_move(self, current_pos: Position, board: Board) -> Tuple[int, int]:
    """Dapatkan gerakan eksplorasi cerdas alih-alih acak"""
    # Coba bergerak ke area yang kurang dieksplorasi (pusat papan)
    board_center_x = board.width // 2 if hasattr(board, 'width') else 10
    board_center_y = board.height // 2 if hasattr(board, 'height') else 10

    # Bergerak ke pusat dengan sedikit keacakan
    if random.random() < 0.7: # 70% kemungkinan bergerak ke pusat

```

```

        delta_x, delta_y = get_direction(
            current_pos.x,
            current_pos.y,
            board_center_x,
            board_center_y,
        )
        return delta_x, delta_y
    else:
        return self.get_safe_random_move()

def get_safe_random_move(self) -> Tuple[int, int]:
    """Dapatkan gerakan acak"""
    return random.choice(self.directions)

def manhattan_distance(self, pos1: Position, pos2: Position) -> int:
    """Hitung jarak Manhattan antara dua posisi"""
    return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

```

4.1.2 Penjelasan Alur Program

GreedyDiamondBot adalah algoritma bot yang dirancang untuk mengumpulkan diamond dengan pendekatan strategi greedy yang dilengkapi sistem prioritas bertingkat. Fokus utamanya adalah mengoptimalkan perolehan diamond secara efisien, sekaligus menjaga jarak aman dari musuh dan memastikan bot dapat kembali ke base dengan selamat.

Prioritas Pertama: Menghindari Musuh

Langkah awal yang dilakukan bot adalah mendeteksi keberadaan musuh dalam radius berbahaya, yaitu sejauh tiga kotak. Apabila ada musuh yang berada dalam jarak sangat dekat (dua kotak atau kurang), bot akan langsung menghindari dari arah musuh tersebut. Langkah ini menjadi prioritas tertinggi karena mempertahankan keselamatan bot dianggap lebih penting dibandingkan mengejar diamond.

Prioritas Kedua: Kembali ke Base

Jika bot telah mengumpulkan minimal empat diamond, maka ia akan langsung diarahkan kembali ke base untuk menyimpannya. Ambang batas ini diturunkan dari lima menjadi empat untuk mengurangi kemungkinan kehilangan diamond apabila bertemu musuh dalam perjalanan.

Prioritas Ketiga: Keluar dari Base

Apabila bot sedang berada di base namun masih membawa diamond, ia akan diarahkan untuk segera meninggalkan area base. Hal ini bertujuan agar bot tidak hanya diam di tempat dan bisa kembali melakukan penjelajahan demi mengumpulkan diamond tambahan.

Prioritas Keempat: Pengambilan Diamond (Greedy Scoring)

Dalam situasi aman, bot akan mencari semua diamond yang tersedia dan memilih target terbaik berdasarkan perhitungan skor: rasio antara jarak dan nilai diamond ($\text{jarak} / \text{nilai}$). Diamond dengan nilai tinggi dan posisi yang dekat akan lebih diprioritaskan. Selain itu, bot juga memperhitungkan faktor risiko — diamond yang berada terlalu dekat dengan musuh akan dikenai penalti skor.

Prioritas Kelima: Kembali ke Base (Alternatif)

Jika bot sudah mengantongi beberapa diamond namun tidak menemukan target yang layak untuk dikejar, maka keputusan terbaik berikutnya adalah kembali ke base. Dengan demikian, diamond yang telah dikumpulkan bisa diamankan daripada hilang karena diserang musuh.

Prioritas Keenam: Eksplorasi Terarah

Saat tidak ada target jelas, bot akan tetap aktif menjelajah. Untuk menghindari eksplorasi acak yang tidak efektif, bot diarahkan untuk lebih sering menuju pusat peta (sekitar 70% peluang), sementara 30% sisanya bergerak secara acak. Ini karena area pusat peta cenderung memiliki distribusi diamond yang lebih banyak.

Optimalisasi Efisiensi

Dalam penghitungan jarak, algoritma menggunakan Manhattan distance karena lebih ringan secara komputasi dibandingkan Euclidean distance. Bot juga dilengkapi dengan mekanisme exception handling agar tidak mengalami crash saat terjadi error, serta memiliki cadangan strategi berupa gerakan random yang tetap berada dalam batas aman.

Keunggulan Utama

Strategi ini menggabungkan respons cepat terhadap ancaman, efisiensi tinggi dalam memilih target, serta keseimbangan antara ambisi mengumpulkan diamond dan perlindungan dari serangan musuh. Bot tidak hanya serakah, tetapi juga cerdas dan adaptif dalam menghadapi situasi dinamis di papan permainan.

4.2 Struktur Data yang Digunakan

Bot ini dirancang dengan menyimpan sejumlah atribut internal yang berperan penting dalam mempercepat proses pengambilan keputusan saat permainan berlangsung. Salah satu atribut utama adalah **directions**, yaitu daftar empat arah dasar (atas, bawah, kiri, kanan) yang menjadi acuan arah gerak bot. Selain itu, bot memanfaatkan variabel bertipe **Optional** untuk menyimpan informasi target sementara, seperti posisi diamond yang ingin diambil atau titik tujuan lainnya. Bot juga memiliki konstanta berupa bilangan bulat yang digunakan sebagai parameter seperti radius pengawasan terhadap keberadaan musuh di sekitar.

Dari sisi input, bot menerima dua objek inti dari game engine, yaitu **GameObject** yang mewakili dirinya sendiri beserta propertinya (seperti lokasi, jumlah diamond, dan posisi base), serta **Board** yang berisi seluruh informasi terkait keadaan papan permainan, mulai dari letak musuh, diamond, hingga objek lainnya. Posisi entitas direpresentasikan menggunakan struktur **Position** yang terdiri dari koordinat x dan y, sedangkan objek-objek di papan (seperti diamond, bot lain, maupun teleporter) dibentuk sebagai **GameObject** lengkap dengan ID, posisi, dan atribut tambahan yang dapat diakses dengan mudah.

Untuk memastikan efisiensi saat pemrosesan data, semua diamond dan musuh disusun dalam **List[GameObject]**, yang diperoleh melalui proses penyaringan dari keseluruhan objek di papan.

Hal ini memungkinkan pemrosesan dilakukan secara batch, seperti dalam proses seleksi diamond terbaik atau pengecekan musuh di sekitar bot.

Semua fungsi gerakan dalam algoritma mengembalikan pasangan nilai (**delta_x**, **delta_y**) yang menunjukkan arah gerakan bot dari posisi saat ini menuju target. Format ini dipilih karena sudah sesuai dengan format input yang diharapkan oleh game engine, sehingga integrasi antara bot dan engine berjalan tanpa hambatan tambahan.

Dari sisi keamanan program, penggunaan tipe **Optional** sangat penting untuk menghindari error akibat referensi ke variabel kosong (null), terutama saat bot belum memiliki target yang pasti. Pendekatan ini mendukung penulisan kode yang lebih aman dan eksplisit dalam menangani kasus tanpa target.

Dalam sistem pemilihan diamond terbaik, algoritma menggunakan tipe data **float** dengan nilai awal berupa **infinity** untuk menjaga akurasi perbandingan skor antar diamond. Metode ini memastikan diamond pertama yang valid langsung menjadi kandidat awal, dan diamond berikutnya akan dibandingkan dengan lebih presisi. Skor dihitung dari rasio antara jarak ke diamond dan nilainya, ditambah dengan faktor keamanan dari lokasi diamond tersebut. Semakin rendah skor, semakin layak diamond tersebut untuk dikejar.

Keseluruhan struktur data yang digunakan oleh bot ini disusun secara sederhana namun efektif. Setiap komponennya dibuat untuk memudahkan akses terhadap informasi game, mempercepat proses evaluasi target, dan menjaga agar bot tetap stabil serta adaptif saat menghadapi dinamika permainan yang kompleks dan kompetitif.

4.3 Pengujian Program

4.3.1 Skenario Pengujian

Untuk memastikan bahwa bot mampu beroperasi secara optimal dalam lingkungan permainan yang kompleks dan terus berubah, dilakukan serangkaian pengujian menyeluruh yang mencakup berbagai aspek fungsionalitas dan kinerja algoritma. Setiap pengujian dirancang dengan skenario spesifik dan indikator evaluasi yang terukur guna menilai seberapa efektif dan stabil bot dalam merespons situasi permainan yang beragam.

1. Pengujian Prioritas Keamanan

Pengujian ini difokuskan untuk memastikan bahwa bot benar-benar mengutamakan keselamatan daripada mengejar diamond. Skenario pengujian mencakup keberadaan satu hingga beberapa musuh dalam radius tertentu, serta variasi batas aman jarak musuh. Keberhasilan bot dievaluasi berdasarkan kemampuannya menghindari zona berisiko sebelum terkena serangan atau kehilangan diamond.

2. Pengujian Logika Greedy

Uji ini memverifikasi keakuratan algoritma greedy dalam menentukan diamond target. Pengujian dilakukan dengan mengecek apakah sistem scoring yang digunakan—yakni perbandingan antara jarak dan nilai—berfungsi sebagaimana mestinya, termasuk penerapan penalti untuk diamond yang dekat musuh dan insentif untuk diamond bernilai tinggi. Hasil dari pengujian ini merefleksikan kecerdasan bot dalam memilih target terbaik di tengah variatifnya posisi diamond dan musuh.

3. Pengujian Pengelolaan Inventori

Bagian ini bertujuan menguji apakah bot mampu mengelola jumlah diamond yang dikumpulkan dengan efisien. Pengujian mencakup respons bot saat mencapai atau melewati ambang batas 4 diamond, bagaimana ia memutuskan untuk kembali ke base, serta perilakunya setelah menyimpan diamond. Uji ini juga mengamati keputusan bot saat tidak ada diamond yang layak dikejar—apakah ia tetap menunggu atau kembali.

4. Pengujian Eksplorasi Wilayah

Ketika bot tidak memiliki target yang jelas, pengujian ini mengukur kemampuan eksploratifnya. Fokusnya pada bagaimana bot bergerak menuju pusat peta dan seberapa optimal penggunaan algoritma pathfinding sederhana seperti Manhattan distance. Evaluasi dilakukan untuk memastikan bot tetap aktif dan bergerak dengan arah yang logis meskipun dalam kondisi tanpa target.

5. Pengujian Toleransi Kesalahan

Aspek ini menguji ketangguhan algoritma ketika dihadapkan pada kondisi tak terduga, seperti data yang tidak lengkap, input yang salah, atau error saat runtime. Pengujian bertujuan memastikan bahwa bot dapat merespons dengan strategi cadangan, seperti bergerak secara acak namun aman, tanpa menyebabkan crash atau gangguan perilaku.

6. Pengujian Performa Sistem

Bagian ini mengukur performa teknis algoritma, termasuk waktu respon, efisiensi penggunaan memori, dan kemampuannya beroperasi di papan permainan yang besar dan padat objek. Tujuannya adalah memastikan bahwa algoritma tetap ringan dan cepat, sehingga dapat digunakan dalam lingkungan *real-time* tanpa menyebabkan keterlambatan atau beban sistem berlebih.

7. Pengujian Kompetitif Antar bots

Uji coba dilakukan dalam situasi permainan yang kompetitif, di mana beberapa bot bersaing dalam kondisi terbatasnya sumber daya. Termasuk di dalamnya adalah fase permainan akhir (late

game), keterbatasan diamond, dan interaksi langsung antar bot. Pengujian ini menilai kemampuan adaptasi algoritma dalam menghadapi tekanan tinggi dan situasi permainan yang intens.

Masing-masing pengujian didukung oleh metrik evaluasi khusus, seperti tingkat keberhasilan (success rate), efisiensi (efficiency score), tingkat keselamatan (safety score), dan waktu respon (response time). Dengan pendekatan evaluasi yang sistematis dan menyeluruh ini, dapat disimpulkan bahwa *GreedyDiamondBot* tidak hanya unggul dalam pengambilan keputusan secara logis, tetapi juga tangguh dan andal dalam menghadapi tantangan permainan nyata.

4.3.2 Hasil Pengujian dan Analisis

1. Strategi Efisien

Strategi efisien terbukti menjadi yang paling konsisten selama pengujian, khususnya pada dua papan awal dengan capaian skor tertinggi 15 dan 12. Pendekatan yang menggabungkan nilai diamond, tingkat keamanan, dan jarak tempuh melalui rumus $(\text{value} \times \text{safety}) / \text{distance}$ memungkinkan bot memilih target yang tidak hanya menguntungkan secara poin tetapi juga aman untuk dikejar. Strategi ini sangat adaptif dalam situasi papan yang kompleks, seperti ketika diamond tersebar tidak beraturan atau berada di dekat musuh. Meskipun demikian, dalam peta yang penuh diamond dan minim ancaman, strategi ini sedikit tertinggal karena sifatnya yang cenderung berhati-hati.

2. Strategi Rasio (value / distance)

Strategi ini memprioritaskan efisiensi murni dengan menghitung rasio antara nilai diamond dan jaraknya dari posisi bot. Pendekatan ini menunjukkan performa kuat pada papan ketiga dan kelima, terutama saat kondisi permainan relatif aman dan diamond tersebar cukup merata. Kelebihannya terletak pada kemampuan bot dalam memilih target dengan cepat dan tepat berdasarkan rasio keuntungan tertinggi. Namun, karena tidak mempertimbangkan faktor keamanan, bot menjadi rentan terhadap ancaman musuh saat berada di area berisiko tinggi.

3. Strategi Safe

Strategi safe menitikberatkan pada keselamatan bot dengan menjauhi musuh dan menghindari area berbahaya. Pendekatan ini paling efektif pada papan keempat, di mana kemungkinan besar banyak diamond berada di zona rawan. Bot yang menerapkan strategi ini mampu bertahan lebih lama dan mengamankan diamond tanpa banyak risiko. Meskipun sangat andal dalam permainan yang penuh tekanan dari musuh, strategi ini dapat kehilangan keunggulan dalam kondisi permainan yang lebih terbuka karena terlalu menghindari risiko.

4. Strategi Dekat

Strategi ini berfokus pada pengambilan diamond yang berada paling dekat dengan posisi bot, tanpa memperhitungkan nilai maupun risiko. Efektivitas strategi ini terlihat jelas pada papan ketiga, ketika diamond berkelompok dalam jarak pendek. Bot dapat bergerak cepat antar target,

sehingga efisien dari segi waktu. Namun, pendekatan ini cenderung mengabaikan potensi nilai poin dan bahaya dari musuh, yang menyebabkan kinerjanya menurun pada papan yang luas atau dalam kondisi permainan dengan banyak musuh.

5. Strategi Tinggi (Highest Value)

Pendekatan ini mengarahkan bot untuk selalu mengejar diamond dengan nilai tertinggi, terutama diamond merah, tanpa mempertimbangkan jarak maupun tingkat risiko. Meskipun secara teori dapat menghasilkan skor besar, dalam praktiknya strategi ini seringkali tidak efektif. Dalam pengujian, strategi ini gagal mencetak skor pada dua papan, menunjukkan bahwa terlalu fokus pada satu target bernilai tinggi bisa merugikan ketika target tersebut jauh atau berada di area yang sulit dijangkau. Strategi ini kurang fleksibel dan kerap membuang waktu jika situasi tidak mendukung

Laga	Pemenang	Skor Tertinggi	Catatan
1	Efisien	15	Strategi efisien unggul karena berhasil mengelola jarak, nilai, dan keamanan.
2	Efisien	12	Meski distribusi diamond minim, efisien tetap adaptif.
3	Rasio dan Dekat	10	Rasio unggul karena distribusi diamond padat dan merata.
4	Safe	15	Strategi aman menang karena banyak diamond dekat area musuh.
5	Rasio	10	Pemilihan berdasarkan rasio value/distance lebih efisien di peta luas.

Dari seluruh strategi yang diuji, strategi **greedy by efisiensi** terbukti paling seimbang dan adaptif terhadap berbagai skenario permainan. Pendekatan ini memungkinkan bot untuk mengumpulkan diamond dengan mempertimbangkan nilai, jarak, dan tingkat keamanan secara bersamaan. Strategi lain seperti **rasio** dan **safe** juga menunjukkan performa baik dalam kondisi tertentu, tetapi tidak sefleksibel efisien dalam menghadapi perubahan situasi di papan permainan. Oleh karena itu, strategi efisiensi dinilai sebagai pilihan utama dalam menciptakan bot yang kompetitif dan cerdas.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari rangkaian pengujian terhadap lima strategi greedy yang diterapkan — yaitu Efisien, Rasio, Safe, Dekat, dan Tinggi — dapat disimpulkan bahwa strategi Greedy by Efisiensi menjadi pendekatan yang paling konsisten, fleksibel, dan unggul dalam berbagai konfigurasi permainan. Pendekatan ini mengintegrasikan tiga elemen kunci: nilai diamond, jarak ke target, dan faktor keamanan dari musuh, yang secara bersamaan menghasilkan keputusan yang proporsional antara mengambil risiko dan mengamankan poin. Keunggulan strategi ini terbukti dari dominasinya di papan pertama dan kedua, serta performanya yang tetap kompetitif pada pengujian lainnya.

Sementara itu, strategi Rasio (perbandingan nilai terhadap jarak) dan Safe juga menunjukkan efektivitas masing-masing pada kondisi tertentu. Rasio tampil kuat di papan dengan distribusi diamond yang merata dan risiko musuh yang rendah, sedangkan Safe sangat cocok untuk kondisi penuh ancaman, di mana musuh berada dalam jangkauan yang membahayakan. Kedua strategi ini ideal digunakan sebagai opsi tambahan atau penyesuaian tergantung situasi permainan.

Di sisi lain, strategi Dekat dan Tinggi memiliki keterbatasan yang cukup jelas. Strategi Dekat hanya mengutamakan jarak tanpa mempertimbangkan nilai maupun potensi bahaya, membuatnya kurang efisien dalam jangka panjang. Strategi Tinggi, yang hanya mengejar diamond bernilai besar, terbukti kurang efektif karena tidak memperhitungkan jarak dan risiko, sehingga bot kerap gagal meraih skor optimal.

Pengujian dilakukan dalam berbagai variasi situasi, seperti distribusi diamond yang acak atau terkonsentrasi, keberadaan musuh yang dinamis, serta kondisi awal hingga akhir permainan. Indikator evaluasi seperti skor total, rasio keberhasilan, dan efisiensi pergerakan semakin memperkuat temuan bahwa strategi efisien unggul dalam pengambilan keputusan dan penyesuaian terhadap situasi yang kompleks.

Dengan mempertimbangkan keseluruhan hasil, strategi Greedy by Efisiensi dipilih sebagai pendekatan utama karena terbukti mampu mengakomodasi kompleksitas permainan dengan baik. Sifat strateginya yang logis dan adaptif menjadikannya pondasi ideal dalam membangun bot yang kompetitif dan cerdas di lingkungan permainan Diamonds.

5.2 Saran

Berdasarkan analisis mendalam terhadap performa strategi Greedy by Efisiensi yang telah terbukti unggul, beberapa saran pengembangan dapat diimplementasikan untuk memaksimalkan potensi bot. Pertama, implementasikan sistem hybrid intelligent yang menggunakan strategi Efisien sebagai core strategy namun dilengkapi dengan conditional switching mechanism. Bot dapat secara otomatis beralih ke strategi Safe ketika density musuh meningkat atau ke strategi Rasio pada early game phase ketika eksplorasi masih menjadi prioritas utama.

Saran kedua adalah pengembangan adaptive weighting system yang dapat menyesuaikan bobot ketiga faktor kunci (nilai diamond, jarak, keamanan) secara real-time berdasarkan game state. Implementasi machine learning algorithm untuk parameter optimization akan memungkinkan bot melakukan self-tuning dan meningkatkan decision accuracy dari waktu ke waktu. Tambahkan juga predictive analytics untuk mengantisipasi movement pattern musuh dan resource distribution, sehingga bot dapat melakukan proactive planning dibanding hanya reactive response.

Dari aspek teknis, disarankan untuk membangun modular architecture dengan clean separation antara strategy engine, execution layer, dan monitoring system. Implementasikan comprehensive logging dan real-time performance analytics untuk continuous improvement. Eksplorasi advanced algorithms seperti Monte Carlo Tree Search untuk complex scenario planning atau reinforcement learning untuk long-term strategy optimization juga menjadi langkah penelitian yang strategic. Terakhir, pertimbangkan pengembangan ensemble approach yang menggabungkan multiple AI techniques untuk menciptakan robust decision-making system yang dapat beradaptasi dengan berbagai kompleksitas permainan yang mungkin muncul di masa depan.

LAMPIRAN

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

B. Video Penjelasan ([link GDrive](#))

DAFTAR PUSTAKA

- [1] R. Munir, Algoritma Greedy (Bagian 1), Bahan Kuliah IF2211 Strategi Algoritma, Sekolah Teknik Elektro dan Informatika, ITB, 2021.
- [2] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, 1998.

