IMPLEMENTASI ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN DIAMONDS

Tugas Besar

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

Oleh: Kelompok 3 (Complexity)

Nashrullah Fathul Qoriib	122140162
Atalie Salsabila	123140027
Eka Putri Azhari Ritonga	123140028

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

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Gambar 1.1 Permainan programming challenge Diamonds

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

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

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan game engine dan membuat bot dari bot starter pack yang telah tersedia pada pranala berikut.

- **Game engine :**
  https://github.com/haziqam/tubes1-IF2211-game-engine.git
- **Bot starter pack :**
  https://github.com/haziqam/tubes1-IF2211-bot-starter-pack.git

Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds

Gambar 1.2 Komponen 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**

Gambar 1.3 Komponen Reset Button

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

3. **Teleporters**

Gambar 1.4 Komponen Teleporters

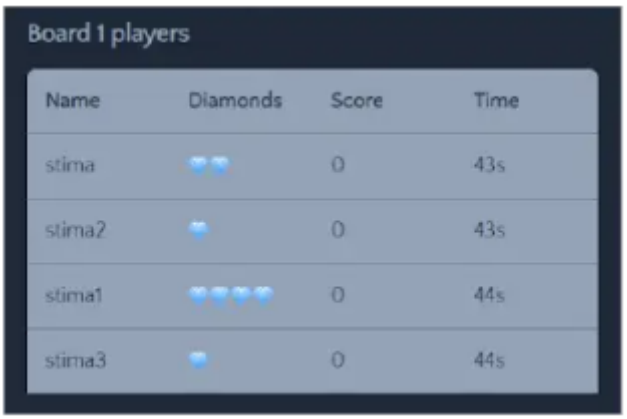
Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

4. **Bots and Bases**

Gambar 1.4 Komponen 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**



Gambar 1.5 Iventory Bots

Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga

sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.

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

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

Spesifikasi permainan yang digunakan pada tugas besar ini disesuaikan dengan spesifikasi yang disediakan oleh game engine Diamonds. Beberapa spesifikasi bot permainan Diamonds adalah sebagai berikut:

1. Buatlah program sederhana dalam bahasa **Python** yang mengimplementasikan ***algoritma Greedy*** pada *bot* permainan Diamonds dengan tujuan memenangkan permainan.
2. Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.
3. Strategi *greedy* yang diimplementasikan setiap kelompok harus dikaitkan dengan fungsi objektif dari permainan ini, yaitu memenangkan permainan dengan memperoleh *diamond* sebanyak banyak nya dan jangan sampai *diamond* tersebut diambil oleh bot lain. Buatlah strategi *greedy* terbaik, karena setiap bot dari masing-masing kelompok akan diadu dalam kompetisi Tubes 1.

4. Strategi *greedy* yang kelompok anda buat harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas yang bermacam macam dalam menyusun strategi *greedy* untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada *game engine* yang telah disebutkan diatas serta dapat dikompetisikan dengan bot dari kelompok lain.
5. Program harus mengandung komentar yang jelas, dan untuk setiap strategi *greedy* yang disebutkan, harus dilengkapi dengan kode sumber yang dibuat.
6. Mahasiswa dilarang menggunakan kode program yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, diperbolehkan untuk belajar dari program yang sudah ada.
7. Mahasiswa dianggap sudah melihat dokumentasi dari *game engine,* sehingga tidak terjadi kesalahpahaman spesifikasi antara mahasiswa dan asisten.
8. BONUS (maks 10): Membuat video tentang aplikasi *greedy* pada bot serta simulasinya pada game kemudian mengunggahnya di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll.
9. Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan “jawaban”.
10. Terdapat juga demo dari program yang telah dibuat. Pengumuman tentang demo menunggu pemberitahuan lebih lanjut dari asisten.
11. Bot yang telah dibuat akan dikompetisikan dengan kelompok lain dan disaksikan oleh seluruh peserta kuliah. Terdapat hadiah menarik bagi kelompok yang memenangkan kompetisi.

BAB II
LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* adalah metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi langkah. Persoalan optimasi yang akan digunakan pada bot permainan Etimo *Diamonds* adalah mengambil *diamonds* sebanyak-banyaknya untuk dikumpulkan ke dalam *base* bots. Algoritma greedy adalah algoritma yang untuk membentuk solusi langkah langkah perlangkah. Pada setiap langkah tersebut akan dipilih keputusan yang paling optimal. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil, dan keputusan tersebut tidak dapat diubah lagi pada langkah selanjutnya

Permasalahan yang diselesaikan menggunakan algoritma *greedy* dapat dipecah menjadi berbagai elemen-elemen seperti berikut:

1. **Himpunan kandidat**
   Himpunan ini berisi kandidat yang akan dipilih pada setiap langkah (misal: simpul/sisi di dalam graf, *tools*, *task*, koin, objek, karakter, dsb).

2. **Himpunan solusi**
   Kumpulan elemen yang dipilih dari himpunan kandidat untuk membentuk solusi akhir.

3. **Fungsi solusi**
   Fungsi yang mengevaluasi solusi yang berada pada himpunan solusi dengan tujuan untuk menentukan solusi yang optimum.

4. **Fungsi seleksi (*selection function*)**
   Fungsi yang akan memilih kandidat berdasarkan suatu strategi atau ketentuan tertentu. Fungsi ini merupakan suatu pendekatan *heuristic*.

5. **Fungsi kelayakan (*feasible*)**
   Fungsi akan memeriksa kandidat apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).

6. **Fungsi obyektif**
   Fungsi ini bertujuan untuk menjelaskan tujuan dari algoritma seperti memaksimumkan atau meminimumkan.

Algoritma *greedy* tidak selalu optimal dalam memecahkan masalah. Perlu dilakukan analisa yang mendalam dalam menerapkan algoritma *greedy* agar solusi dari *greedy* dapat mendekati optimum atau bahkan optimum. Hal penting dalam menerapkan algoritma *greedy* adalah dengan cara memperhatikan input *counterexample* dan mencoba untuk melakukan suatu penanganan khusus untuk *counterexample*. *Counterexample* harus diuji dan ditangani agar setidaknya program yang kita rancang tidak memberikan solusi yang jauh dari kata optimal.

Dalam permasalahan *knapsack*, perlu dilakukan berbagai alternatif solusi seperti *greed by weight*, *greed by profit*, dan *greed by density*. Dari berbagai alternatif fungsi solusi ini akan dilakukan suatu seleksi berdasarkan output yang memberikan hasil maksimum. Dengan mempertimbangkan berbagai kasus, solusi dari algoritma *greedy* dapat semakin mendekati solusi optimal atau bahkan akan optimal terus menerus.

2.2 Game Engine Diamonds

Game engine dari permainan Diamonds dapat ditemukan pada tautan https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0. Game engine ini telah dipersiapkan sehingga kita hanya perlu melakukan konfigurasi untuk menggunakannya. Ada beberapa *dependencies* yang digunakan pada game engine ini seperti:

1. Node.js
2. Docker desktop

Game engine Diamonds tidak memanfaatkan engine compose, sehingga diperlukan Docker untuk memproses serta menjalankan seluruh komponen yang ada. Docker juga berperan dalam memastikan aplikasi berjalan secara mandiri dan stabil di lingkungan mesin lokal.

2.3 Struktur Game Engine

Mengetahui komponen-komponen dalam game engine Diamonds sangatlah penting, terutama untuk mendukung proses pengembangan bot dalam permainan tersebut. Berdasarkan tautan *releases* yang dibahas pada subbab 2.2., game engine ini memiliki beberapa komponen dengan struktur sebagai berikut:

1. Folder *frontend* – berfungsi sebagai antarmuka pengguna untuk permainan Diamonds.
2. Folder *backend* – berperan dalam pengolahan data dan mengelola folder *types* yang terhubung dengan *frontend*, mencakup berbagai komponen permainan seperti Diamonds, Teleporter, Reset Button, Base, dan juga bot.
3. Folder *type* – digunakan untuk menyimpan *game objects* seperti base, bot, diamond, dummy-bot, teleport, lokasi teleportasi, dan tombol reset

.

2.4 Bot Engine Diamonds

Terdapat dua file utama yang sangat penting dalam direktori bot engine Diamonds, sebagaimana dijelaskan pada tautan *releases* di subbab 2.2. Kedua file ini perlu dipahami karena membantu menjelaskan bagaimana bot engine bekerja, terlepas dari logika pemrograman yang kita buat sendiri. Adapun file penting yang mendukung kerja bot engine ini meliputi:

1. **main.py** – Berperan dalam membantu bot agar dapat terhubung dengan server dalam permainan Diamonds melalui API. File ini juga memungkinkan bot menerima sejumlah argumen dari command line untuk mengatur cara bermainnya.

2. decode.py – Bertugas memproses data dalam bentuk dictionary atau daftar dictionary. File ini mengubah format kunci pada data menjadi snake case menggunakan fungsi _keys_to_snake_case(), serta melakukan rekursi jika terdapat dictionary bersarang (nested).

Selain kedua file tersebut, bot engine ini juga memiliki folder *game* yang berfungsi sebagai pendukung dan landasan untuk menjalankan logika program yang telah dibuat. Dalam laporan ini akan dibahas file-file penting yang ada di dalam folder *game*, seperti *util* dan juga folder *logic*. Dalam proyek tugas besar ini, peserta diminta untuk mengembangkan logika program menggunakan algoritma *greedy*

2.5 Alur Permainan Diamonds by Etimo

Sebelum mengembangkan bot dan menjalankan game engine Diamonds dari Etimo, perlu menginstal beberapa dependencies, yaitu:

1. **Game Engine**: Node.js, Docker Desktop
2. **Bot Engine**: Python

2.5.1 Menjalankan Game Engine Diamonds by Etimo

Dalam permainan Diamonds dari Etimo, pengguna tidak perlu mengubah konfigurasi atau isi dari folder game engine. Sebelum menjalankannya, pastikan perangkat telah terpasang *Node.js,* Docker Desktop. Setelah semua prasyarat terinstal, ikuti langkah-langkah berikut untuk menjalankan game engine:

1. Unduh file *source code* dalam format .zip dari tautan berikut: https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0
2. Ekstrak file tersebut, lalu buka terminal di dalam folder hasil ekstraksi.
3. Arahkan ke direktori utama proyek:

```
cd tubes1-IF2211-game-engine-v1.1.0
```

4. Lakukan konfigurasi *environment variable* dengan menjalankan skrip berikut:

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

5. Siapkan local database (pastikan Docker Desktop sudah aktif), lalu jalankan perintah berikut di terminal:

```
compose up -d database
```

6. Jalankan skrip berikut untuk sistem operasi Windows:

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

7. Jalankan perintah untuk melakukan *build*:

```
npm run build
```

8. Jalankan perintah untuk memulai aplikasi:

```
npm run start
```

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping* Persoalan ke Dalam Elemen Greedy

Dalam game Etimo Board, bot harus mengambil keputusan untuk memilih diamond mana yang akan diambil. Masalah ini bisa diselesaikan dengan algoritma greedy. Supaya algoritma ini bisa dipakai, kita perlu memetakan elemen-elemen di game ke dalam bagian-bagian greedy sebagai berikut:

1. **Himpunan Kandidat:**
   Semua diamond yang ada di board permainan. Masing-masing punya nilai (point) dan posisi (koordinat).
2. **Himpunan Solusi:**
   Kumpulan diamond yang akhirnya dipilih dan dikunjungi oleh bot.
3. **Fungsi Solusi:**
   Bot akan terus menambahkan diamond ke jalurnya selama masih muat di inventori dan layak diambil.
4. **Fungsi Seleksi:**
   Bot menghitung skor tiap diamond dengan rumus:
   score = points / (distance + 1)
   Lalu pilih diamond dengan skor tertinggi. Tapi bot juga mengecek apakah ada bot lain yang lebih dekat agar tidak bentrok.
5. **Fungsi Kelayakan:**
   Cek apakah diamond masih bisa diambil (misalnya tidak terlalu dekat dengan lawan atau tidak lewat teleport). Bot juga pastikan inventori tidak penuh.
6. **Fungsi Objektif:**
   Tujuan akhirnya adalah dapat poin sebanyak mungkin dalam langkah yang efisien.

3.2 Eksplorasi Alternatif Solusi Greedy

Pada implementasi bot ini, strategi greedy yang digunakan adalah memilih diamond berdasarkan skor hasil pembagian poin dengan jarak **score = points / (distance + 1)**. Namun, sebenarnya ada beberapa alternatif pendekatan greedy lain yang juga bisa digunakan, tergantung prioritas yang ingin dicapai oleh bot. Berikut beberapa contohnya:

1. **Greedy Berdasarkan Jarak Terdekat Saja:**
   Bot hanya fokus memilih diamond yang paling dekat, tanpa mempertimbangkan nilai poinnya. Pendekatan ini bisa lebih cepat karena tidak banyak perhitungan, tapi berisiko mengambil diamond bernilai kecil.

2. **Greedy Berdasarkan Nilai Tertinggi (Point Tertinggi):**
   Bot memilih diamond dengan poin paling besar dulu, tidak peduli jaraknya. Pendekatan ini bisa menghasilkan poin besar, tapi bisa boros langkah dan waktu.

3. **Greedy Kombinasi dengan Area Aman (Safe Zone):**
   Selain mempertimbangkan skor dan posisi lawan, bot hanya bergerak ke arah diamond yang berada di area aman atau minim risiko. Ini bisa digabungkan dengan strategi Safe Random Move jika tidak ada pilihan pasti.

4. **Greedy Berdasarkan Potensi Blok Lawan:**
   Bot bisa memilih diamond yang tidak hanya menguntungkan, tapi juga bisa mencegah lawan mendapatkan poin besar. Ini semacam strategi greedy yang lebih agresif dan kompetitif.

5. **Greedy dengan Prediksi Jalur:**
   Bot menghitung beberapa langkah ke depan (meskipun tidak sampai full A*) dan tetap memilih langkah lokal terbaik, tapi mempertimbangkan kemungkinan rute selanjutnya.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Bot kami menggunakan beberapa strategi greedy yang sudah kami analisis dari segi efisiensi dan efektivitasnya:

1. **Greedy Diamond Targeting**
   Bot memilih diamond berdasarkan skor yang dihitung dari nilai poin dibagi jarak ke diamond tersebut. Jadi, diamond yang dekat dan bernilai tinggi akan lebih diutamakan. Strategi ini cukup efisien karena sederhana dan cepat dalam menentukan target.
2. **Smart Claiming**

Bot akan menghindari diamond yang terlalu dekat dengan bot lain. Tujuannya agar tidak berebut dan langkah tidak sia-sia. Strategi ini membuat bot jadi lebih pintar dan tidak mudah kalah bersaing.

3. **Return to Base**

   Kalau inventori bot sudah penuh, maka dia langsung kembali ke base untuk menyimpan diamond. Ini membuat bot tidak buang-buang waktu dan langsung mengamankan poin.

4. **Avoid Teleport & Safe Random Move**

   Bot juga menghindari posisi teleport karena bisa membuatnya berpindah tempat secara acak. Kalau tidak ada langkah yang terbaik, bot akan bergerak secara acak tapi tetap aman.

3.4 Strategi Greedy yang Dipilih

Strategi utama yang kami pilih dan implementasikan di program ini adalah:

Greedy Diamond Targeting, yaitu memilih diamond terbaik berdasarkan rumus:

score = poin / (jarak + 1)

Alasan kami memilih strategi ini:

- Sederhana dan cepat diproses.
- Bisa menyeimbangkan antara nilai diamond dan jaraknya.
- Cocok untuk kondisi permainan yang berubah-ubah.
- Tidak ribet diimplementasikan dalam kode.

Strategi ini kami anggap paling seimbang antara hasil dengan kecepatan pengambilan keputusan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
# Nasrhrullah Fatul Qoriib (122140162)
# Atalie Salsabila (123140027)
# Eka Putri Azhari Ritonga (123140028)
# Kelompok: Complexity

import random
from typing import Optional, Tuple
from utils.models import Board, Bot, Position, GameObject

# Fungsi untuk menentukan langkah berikutnya
def get_next_move(board: Board, my_bot_data: Bot) -> Optional[Tuple[str, int]]:
    # Ambil objek bot dari board
    my_bot: GameObject = board.get_bot(my_bot_data)

    # Tidak bisa bergerak jika bot tidak ditemukan
    if not my_bot or not my_bot.position:
        return None

    # Posisi dan properti dasar dari bot
    my_pos = my_bot.position
    base_pos = my_bot.properties.base
    inventory = my_bot.properties.diamonds or 0
    max_inventory = my_bot.properties.inventory_size or 5

    # Hindari semua posisi teleport
    teleport_positions = {
        obj.position for obj in board.game_objects
        if obj.position and obj.type == "TeleportGameObject"
    }

    # === PRIORITAS 0: Pulang ke base jika inventori penuh ===
    if inventory >= max_inventory:
```

```
        if base_pos:
            if not positions_equal(my_pos, base_pos):
                direction = direction_towards(my_pos, base_pos, teleport_positions)
                if direction:
                    return direction, -1  # Menuju base
                else:
                    return random_move(my_pos, teleport_positions)  # Alternatif jika terhalang
            else:
                # Sudah di base tapi belum kosong, mungkin perlu waktu/trigger
                # Lakukan gerakan acak kecil untuk 'bangunkan' bot
                return random_move(my_pos, teleport_positions)
        return None

    # === PRIORITAS 1: Cari diamond terbaik yang kita lebih dekat dari bot lain ===
    target = find_best_diamond(
        board, my_pos, inventory, max_inventory, teleport_positions,
        prefer_closest=True, my_name=my_bot.properties.name
    )

    # === PRIORITAS 2: Jika tidak ada, ambil diamond terbaik tanpa mempertimbangkan
kedekatan bot lain ===
    if not target:
        target = find_best_diamond(
            board, my_pos, inventory, max_inventory, teleport_positions,
            prefer_closest=False
        )

    # === PRIORITAS 3: Jika tetap tidak ada, lakukan gerakan acak (asal tidak masuk teleport)
===
    if target:
        direction, target_id = target
        return direction, target_id

    return random_move(my_pos, teleport_positions)  # -2 menandakan random step

# Fungsi untuk mencari diamond terbaik
def find_best_diamond(board, my_pos, inventory, max_inventory, avoid, prefer_closest=True,
my_name=None):
```

```
# Inisialisasi variabel untuk menyimpan diamond terbaik
best_score = -1
target_pos = None
target_id = None

# Iterasi semua objek di papan
for obj in board.game_objects:
    # Lewati jika bukan diamond atau tidak memiliki posisi
    if obj.type != "DiamondGameObject" or not obj.position:
        continue

    pos = obj.position
    points = obj.properties.points if obj.properties else None

    # Lewati jika diamond tidak memiliki poin atau poin tidak valid
    if points is None or points <= 0:
        continue

    # Lewati jika diamond tidak muat di inventori
    if inventory + points > max_inventory:
        continue

    # Lewati jika sudah di posisi diamond
    if positions_equal(pos, my_pos):
        continue

    # Hitung jarak bot kita ke diamond ini
    my_dist = manhattan_distance(my_pos, pos)

    # === Cek apakah kita lebih dekat dari bot lain ===
    if prefer_closest and my_name:
        # Cek setiap bot lain
        for bot in board.bots:
            if not bot.position or bot.properties.name == my_name:
                continue
            # Jika ada bot lain lebih dekat, diamond ini tidak dipilih
            if manhattan_distance(bot.position, pos) < my_dist:
                break
```

```
            else:
                # Tidak ada bot lain lebih dekat: diamond ini kandidat
                score = points / (my_dist + 1)
                if score > best_score:
                    best_score = score
                    target_pos = pos
                    target_id = obj.id

        # === Alternatif: Ambil diamond terbaik tanpa cek kedekatan bot lain ===
        elif not prefer_closest:
            score = points / (my_dist + 1)
            if score > best_score:
                best_score = score
                target_pos = pos
                target_id = obj.id

    # Jika ditemukan diamond yang valid, kembalikan arahnya dan ID-nya
    if target_pos:
        direction = direction_towards(my_pos, target_pos, avoid)
        if direction:
            return direction, target_id

    # Tidak ada diamond valid ditemukan
    return None

# Memberikan langkah acak
def random_move(pos: Position, avoid: set) -> Optional[Tuple[str, int]]:
    # Membuat dictionary arah dan posisi baru berdasarkan posisi saat ini
    directions = {
        "NORTH": Position(pos.x, pos.y - 1),
        "SOUTH": Position(pos.x, pos.y + 1),
        "WEST": Position(pos.x - 1, pos.y),
        "EAST": Position(pos.x + 1, pos.y),
    }

    # Menyaring hanya arah yang tidak menuju ke posisi yang ingin dihindari
    valid_moves = [
        direction for direction, new_pos in directions.items()
```

```
        if new_pos not in avoid
    ]

    # Jika ada gerakan yang valid, pilih satu secara acak dan kembalikan dengan nilai -2
    if valid_moves:
        return random.choice(valid_moves), -2

    # Jika tidak ada gerakan yang valid, kembalikan None
    return None

# Menghitung dan mengembalikan jarak Manhattan antara dua posisi
def manhattan_distance(p1: Position, p2: Position) -> int:
    return abs(p1.x - p2.x) + abs(p1.y - p2.y)

# Mengembalikan True jika kedua posisi memiliki koordinat x dan y yang sama
def positions_equal(p1: Position, p2: Position) -> bool:
    return p1.x == p2.x and p1.y == p2.y

# Fungsi untuk menentukan langkah ke tujuan
def direction_towards(start: Position, goal: Position, avoid: Optional[set] = None) -> str:
    # Hitung selisih koordinat antara posisi tujuan dan posisi awal
    dx = goal.x - start.x
    dy = goal.y - start.y

    # Jika 'avoid' tidak diberikan, gunakan set kosong
    avoid = avoid or set()

    # Daftar kandidat arah jika semua arah utama diblokir
    candidates = []

    # Tentukan arah utama berdasarkan mana yang lebih dominan: horizontal (dx) atau vertikal (dy)
    if abs(dx) >= abs(dy):
        # Prioritaskan ke arah horizontal (EAST/WEST) terlebih dahulu
        if dx > 0 and not is_blocked(start.x + 1, start.y, avoid):
            return "EAST"
        elif dx < 0 and not is_blocked(start.x - 1, start.y, avoid):
            return "WEST"
```

```
    # Jika tidak bisa ke horizontal, coba ke vertikal (SOUTH/NORTH)
    if dy > 0 and not is_blocked(start.x, start.y + 1, avoid):
        return "SOUTH"
    elif dy < 0 and not is_blocked(start.x, start.y - 1, avoid):
        return "NORTH"
else:
    # Prioritaskan ke arah vertikal (SOUTH/NORTH) terlebih dahulu
    if dy > 0 and not is_blocked(start.x, start.y + 1, avoid):
        return "SOUTH"
    elif dy < 0 and not is_blocked(start.x, start.y - 1, avoid):
        return "NORTH"

    # Jika tidak bisa ke vertikal, coba ke horizontal (EAST/WEST)
    if dx > 0 and not is_blocked(start.x + 1, start.y, avoid):
        return "EAST"
    elif dx < 0 and not is_blocked(start.x - 1, start.y, avoid):
        return "WEST"

# Semua arah utama diblokir, buat daftar semua arah alternatif
directions = {
    "EAST": Position(start.x + 1, start.y),
    "WEST": Position(start.x - 1, start.y),
    "SOUTH": Position(start.x, start.y + 1),
    "NORTH": Position(start.x, start.y - 1),
}

# Tambahkan arah yang tidak ada di posisi yang dihindari ke dalam kandidat
for direction, pos in directions.items():
    if pos not in avoid:
        candidates.append(direction)

# Jika semua arah terblokir, tetap tambahkan semua arah sebagai kandidat (meski berisiko)
if not candidates:
    candidates = list(directions.keys())

# Pilih dan kembalikan satu arah secara acak dari kandidat yang tersedia
return random.choice(candidates)
```

```
# Fungsi untuk mengecek apakah posisi tersebut termasuk dalam set 'avoid'
def is_blocked(x: int, y: int, avoid: set) -> bool:
    return Position(x=x, y=y) in avoid
```

2. **Penjelasan Alur Program**

Berikut adalah penjelasan alur kerja bot yang telah diimplementasikan:

1. Inisialisasi Bot

Bot pertama-tama mendapatkan posisi dan properti penting dari objek board, yaitu:

- Posisi saat ini (my_pos)
- Posisi base (base_pos)
- Jumlah diamond yang dimiliki (inventory)
- Kapasitas maksimum (max_inventory)

Bot juga mengidentifikasi posisi-posisi yang harus dihindari, yaitu posisi teleport.

2. Logika Prioritas Langkah Bot (Algoritma Greedy)

Bot menggunakan pendekatan algoritma greedy dengan urutan prioritas sebagai berikut:

Prioritas 0: Pulang ke Base Jika Inventori Penuh

Jika jumlah diamond yang dibawa sudah mencapai kapasitas maksimal, bot akan langsung bergerak menuju base untuk mengosongkan inventori. Jika bot sudah berada di base, namun belum terjadi pengosongan, maka bot akan melakukan gerakan kecil acak untuk memicu proses pembuangan diamond.

Prioritas 1: Cari Diamond Terbaik yang Lebih Dekat dari Bot Lain

Jika inventori belum penuh, bot akan mencari diamond dengan nilai terbaik berdasarkan rasio antara poin dan jaraknya, hanya jika posisi bot lebih dekat dari semua bot lainnya. Perhitungan skor dilakukan dengan rumus score = points / (jarak + 1) Semakin banyak poin dan semakin dekat, maka nilai skor semakin tinggi dan menjadi prioritas utama.

Prioritas 2: Cari Diamond Terbaik Tanpa Memperhatikan Bot Lain

Jika tidak ditemukan diamond yang bisa diambil lebih cepat dari bot lain, maka bot akan memilih diamond terbaik yang ada di papan tanpa memperhatikan posisi bot lain. Skor tetap dihitung menggunakan rumus yang sama.

Prioritas 3: Gerakan Acak

Jika tidak ada diamond yang bisa dijangkau atau tidak ditemukan target yang valid, maka bot akan melakukan langkah acak ke arah yang aman, dengan menghindari posisi teleport.

3. Penentuan Arah Gerak

Bot menggunakan pendekatan algoritma greedy dengan urutan prioritas sebagai berikut:

Fungsi direction_towards digunakan untuk menentukan arah gerak dari posisi saat ini menuju target. Arah ditentukan berdasarkan selisih koordinat horizontal dan vertikal:

- Jika arah dominan adalah horizontal, maka bot akan mencoba ke arah timur atau barat terlebih dahulu.
- Jika arah dominan adalah vertikal, maka bot akan mencoba ke arah utara atau selatan terlebih dahulu.
- Jika semua arah utama terhalang, bot akan memilih salah satu arah alternatif secara acak dari arah yang tidak terblokir.

4. Perhitungan Jarak

Bot menggunakan jarak Manhattan untuk menghitung jarak antara dua posisi:

jarak = |x1 - x2| + |y1 - y2|

5. Eksekusi Gerak

Setelah arah gerakan ditentukan, bot mengirimkan perintah gerak ke server dan mendapatkan keadaan board terbaru. Proses ini terus berulang setiap tick permainan selama bot masih aktif di board.

4.2 Struktur Data yang Digunakan

Dalam implementasi algoritma greedy ini, bot menggunakan struktur data untuk menyimpan informasi penting seperti posisi bot, objek di papan permainan, serta detail dari setiap objek dan bot yang ada. Struktur data ini memungkinkan bot untuk mengambil keputusan berdasarkan informasi keadaan papan secara real-time.

1. **Class dan Objek**
    - **Board**: Menyimpan keadaan papan permainan, termasuk daftar bot (bots) dan objek permainan (game_objects).
    - **Bot**: Menyimpan informasi bot seperti nama (name), email (email), dan ID (id).
    - **GameObject**: Mewakili objek pada papan, seperti diamond, teleport, atau bot lain. Mempunyai atribut position (koordinat), type (jenis objek), dan properties (informasi tambahan).
    - **Position**: Menyimpan koordinat x dan y pada papan permainan.
2. **Struktur Data Tambahan**
    - **List**: Menyimpan data berurutan, seperti daftar objek pada board (game_objects) dan daftar bot (bots).
    - **Set**: Digunakan untuk menyimpan posisi yang harus dihindari, misalnya posisi teleport.
    - **Tuple**: Digunakan untuk menyimpan hasil langkah bot, misalnya (arah, id_target), contoh: ("NORTH", 5).
    - **Dictionary**: Digunakan untuk memetakan arah dengan posisi baru, misalnya: {"NORTH": Position(x, y-1), "SOUTH": Position(x, y+1)}
3. **Fungsi Pendukung**
    - **manhattan_distance**: Menghitung jarak antara dua posisi.
    - **positions_equal**: Mengecek apakah dua posisi sama.
    - **is_blocked**: Mengecek apakah posisi tertentu masuk dalam daftar posisi yang dihindari.

4.3 Pengujian Program

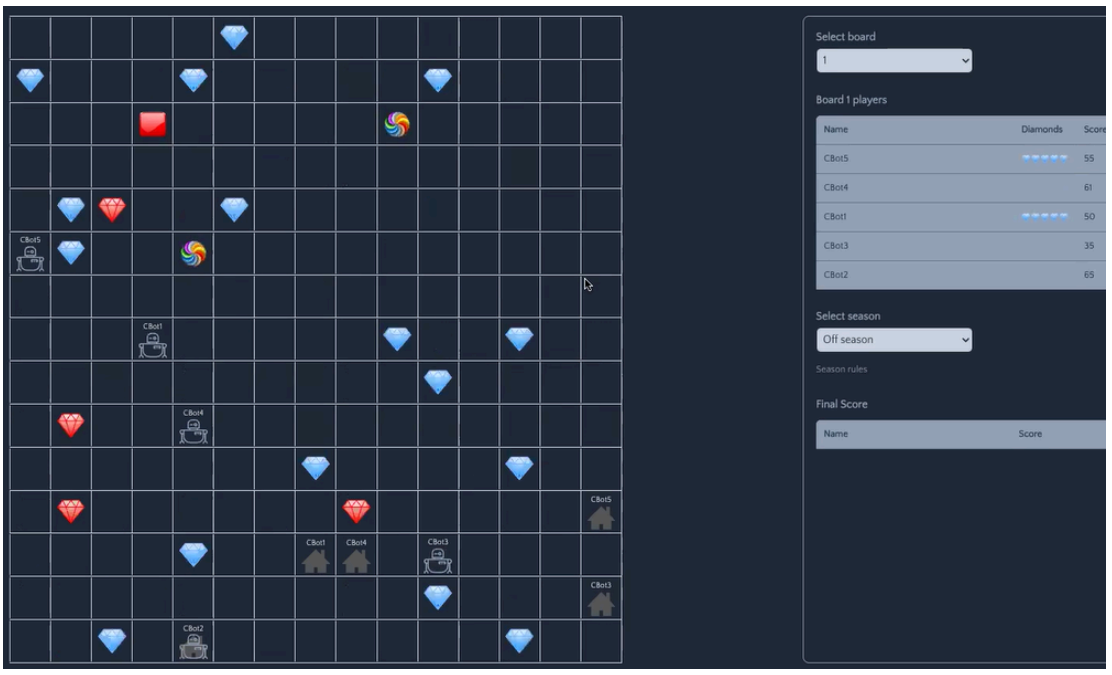
1. Skenario Pengujian

Pengujian dilakukan untuk memastikan bot dapat bergerak dan mengambil keputusan sesuai dengan strategi greedy yang telah diimplementasikan. Beberapa skenario pengujian yang dilakukan antara lain:

1). Bot dapat mendeteksi diamond dengan benar

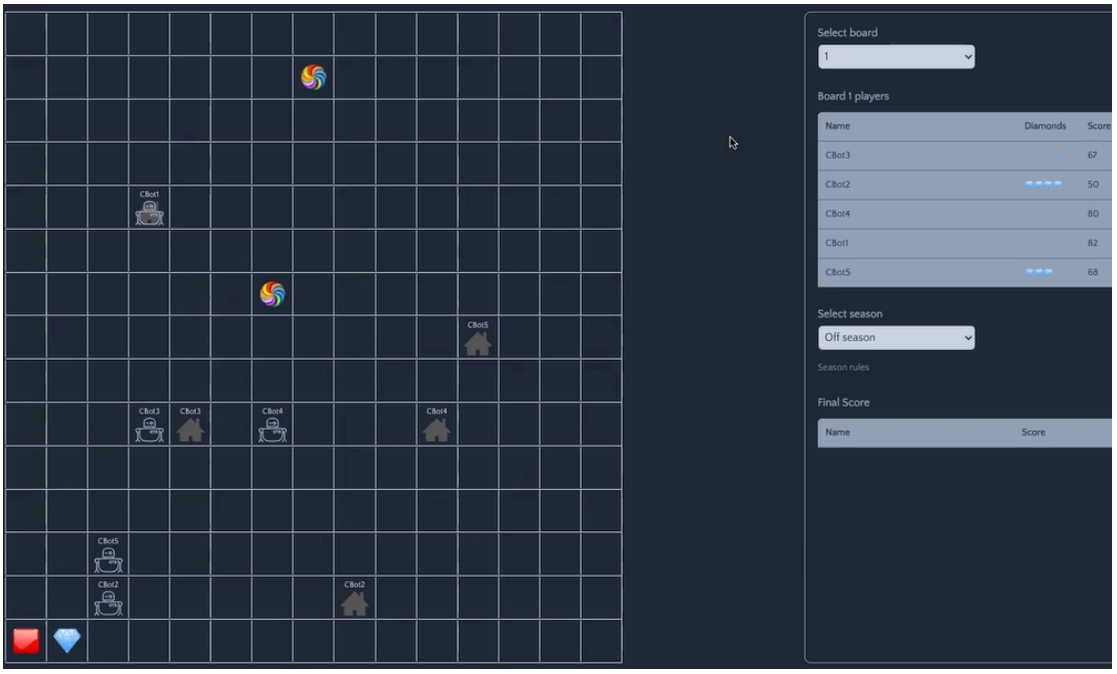
Bot berhasil memilih diamond dengan nilai poin terbaik dan posisi terdekat dibandingkan bot lain.

2). Bot menghindari posisi teleport



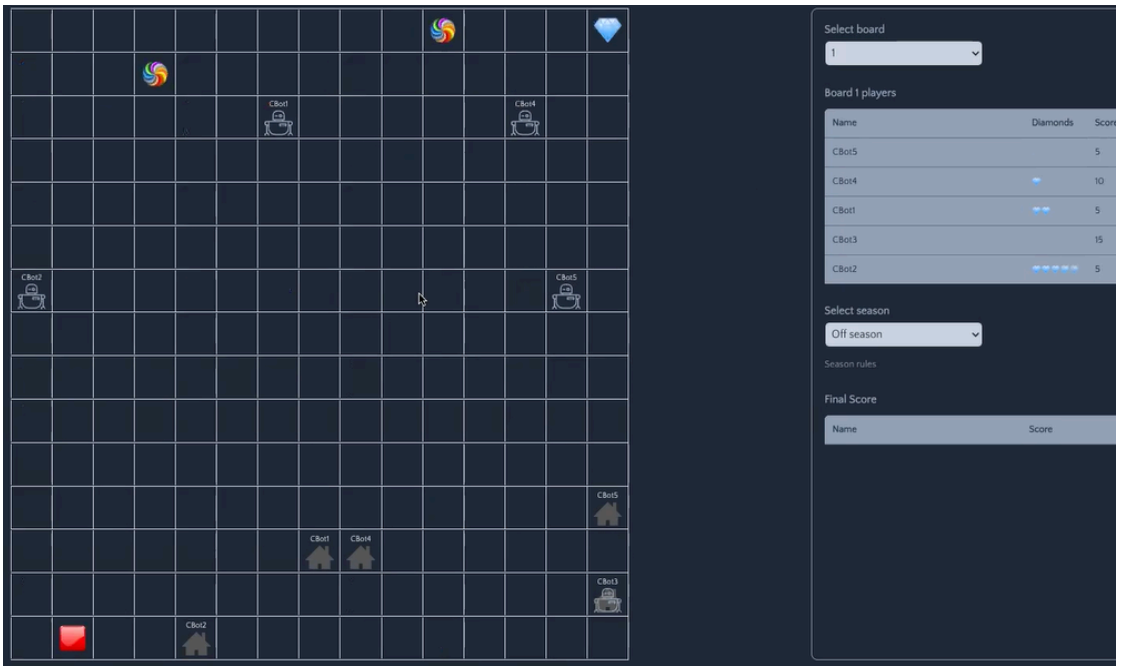
Saat dihadapkan dengan posisi teleport, bot memilih langkah lain untuk menghindari teleport.

3). Bot kembali ke base saat inventory penuh



Saat kapasitas penyimpanan diamond bot sudah penuh, bot bergerak menuju base untuk mengosongkan inventori.

4). Bot bergerak acak jika tidak ada target



Saat tidak ada diamond yang tersedia atau semua diamond tidak bisa diambil, bot melakukan gerakan acak sesuai logika yang telah dibuat.

2. Hasil Pengujian dan Analisis

Berdasarkan pengujian yang telah dilakukan, diperoleh hasil sebagai berikut:

Bot dapat mendeteksi diamond dengan benar
Bot berhasil memilih diamond dengan nilai poin terbaik dan posisi terdekat dibandingkan bot lain.

Bot menghindari posisi teleport
Saat dihadapkan dengan posisi teleport, bot memilih langkah lain untuk menghindari teleport.

Bot kembali ke base saat inventory penuh
Saat kapasitas penyimpanan diamond bot sudah penuh, bot bergerak menuju base untuk mengosongkan inventori.

Bot bergerak acak jika tidak ada target
Saat tidak ada diamond yang tersedia atau semua diamond tidak bisa diambil, bot melakukan gerakan acak sesuai logika yang telah dibuat.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari hasil implementasi dan pengujian program bot permainan Diamonds menggunakan algoritma greedy yaitu:

Algoritma greedy mampu memberikan solusi yang sederhana dan efisien dalam pengambilan keputusan langkah bot. Bot dapat memilih langkah terbaik berdasarkan informasi saat ini tanpa mempertimbangkan dampak jangka panjang. Strategi greedy yang digunakan, yaitu pemilihan diamond dengan nilai skor tertinggi berdasarkan rumus score = points / (jarak + 1), berhasil membuat bot mengutamakan diamond yang memberikan nilai keuntungan maksimal dalam langkah tercepat.

Bot dapat menjalankan logika prioritas dengan baik, yaitu:

- Pulang ke base ketika inventori penuh.
- Memilih diamond terbaik yang lebih dekat dari bot lain.
- Menghindari posisi teleport.
- Melakukan gerakan acak jika tidak ada target yang sesuai.

Hasil pengujian menunjukkan bahwa bot dapat bergerak sesuai strategi yang telah dirancang dan mampu beradaptasi dengan keadaan dinamis di dalam permainan. Penggunaan struktur data yang sesuai, seperti list, set, tuple, dictionary, serta class-class utama (Board, Bot, GameObject, dan Position) mendukung performa program secara keseluruhan. Dengan strategi yang telah diimplementasikan, bot mampu bersaing dengan bot lain dalam permainan untuk memperoleh poin sebanyak mungkin.

5.2 Saran

Beberapa saran yang dapat diberikan untuk pengembangan program bot ke depannya adalah:

1. Menambahkan fitur prediksi langkah lawan atau perhitungan rute lebih dari satu langkah ke depan (lookahead) untuk mengantisipasi pergerakan bot lain.

2. Mengoptimalkan logika pengambilan keputusan dengan mempertimbangkan area yang aman dan risiko bertemu dengan bot lawan, agar strategi greedy lebih adaptif terhadap kondisi permainan.
3. Mengimplementasikan strategi fallback yang lebih cerdas, seperti mempertimbangkan posisi red button atau pola pergerakan diamond yang muncul secara acak.
4. Menambahkan visualisasi yang lebih interaktif agar memudahkan dalam proses debugging dan pemantauan kinerja bot.
5. Melakukan pengujian lebih intensif dengan berbagai skenario permainan dan menghadapi bot dengan strategi yang berbeda, agar dapat mengukur performa secara lebih menyeluruh.

LAMPIRAN

A. **Repository Github (https://github.com/qoriib/ComplexityBot)**
B. **Video Penjelasan (https://drive.google.com/file/d/1M80ZtjWczLWsA2YZ7pkAZFBqeAMmyxkm/view?usp=drivesdk)**
C. **Link Youtube (https://youtu.be/GWEvehJJygc?si=F0S_XdpavBMRgN9a)**

DAFTAR PUSTAKA

[1] S. Oktaviana dan A. Naufal, “Algoritma Greedy untuk Optimalisasi Ruangan dalam Penyusunan Jadwal Perkuliahan,” Jurnal Multinetics, vol. 3, no. 1, pp. 52–56, Mei 2017.

[2] Y. Darnita dan R. Toyib, “Penerapan Algoritma Greedy Dalam Pencarian Jalur Terpendek Pada Instansi-Instansi Penting di Kota Argamakmur Kabupaten Bengkulu Utara,” Jurnal Media Infotama, vol. 15, no. 2, pp. 57–64, Sep. 2019.

[3] R. Munir, "Penggunaan algoritma greedy dalam menyelesaikan permainan," 2008. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2007-2008/Makalah2008/MakalahIF2251-2008-065.pdf.