IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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

Oleh: Kelompok Lightning Yaqueen

Omar Athaya Vito	123140198
Najwa Syahirah Rosyan	123140178
Ahmad Aufamahdi Salam	123140092

Dosen Pengampu: Winda Yulita, M.Cs.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

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

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/releases/tag/v1.1.0

● Bot starter pack :

https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1

Komponen-komponen dari permainan Diamonds antara lain:

Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds

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

2. Red Button/Diamond Button

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

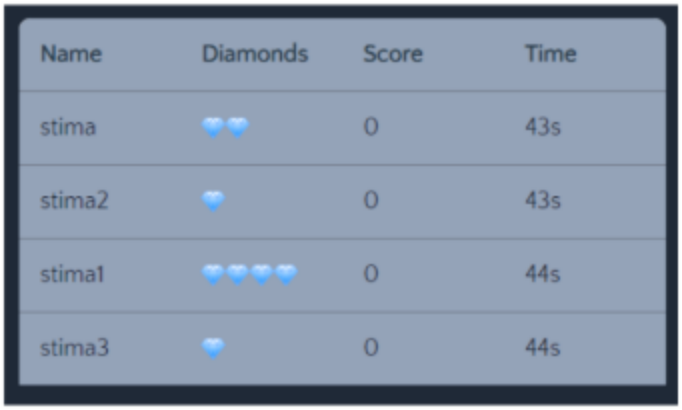
3. Teleporters

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

4. Bots and Bases

Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) bot menjadi kosong.

5. Inventory



Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.

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

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

8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

Spesifikasi Tugas Besar 1

● Buatlah program sederhana dalam bahasa Python yang mengimplementasikan algoritma Greedy pada bot permainan Diamonds dengan tujuan memenangkan permainan.

● Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.

● 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.

● 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.

● Program harus mengandung komentar yang jelas, dan untuk setiap strategi greedy yang disebutkan, harus dilengkapi dengan kode sumber yang dibuat.

● Mahasiswa dilarang menggunakan kode program yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, diperbolehkan untuk belajar dari program yang sudah ada.

● Mahasiswa dianggap sudah melihat dokumentasi dari game engine, sehingga tidak terjadi kesalahpahaman spesifikasi antara mahasiswa dan asisten.

● 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.

● 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”.

● Terdapat juga demo dari program yang telah dibuat. Pengumuman tentang demo menunggu pemberitahuan lebih lanjut dari asisten.

● Bot yang telah dibuat akan dikompetisikan dengan kelompok lain dan disaksikan oleh seluruh peserta kuliah. Terdapat hadiah menarik bagi kelompok yang memenangkan kompetisi.

● Setiap kelompok harap melaporkan nama kelompok dan anggotanya dan diserahkan kepada asisten untuk didata.

● Kelompok yang terindikasi melakukan kecurangan akan diberikan nilai 0 pada tugas besar.

● Program disimpan dalam repository yang bernama Tubes1_NamaKelompok dengan nama kelompok. Berikut merupakan struktur dari isi repository tersebut:

a. Folder src berisi source code.

b. Folder doc berisi laporan tugas besar dengan format NamaKelompok.pdf

c. README untuk tata cara penggunaan yang minimal berisi:

i. Penjelasan singkat algoritma greedy yang diimplementasikan

ii. Requirement program dan instalasi tertentu bila ada

iii. Command atau langkah-langkah dalam meng-compile atau build program

iv. Author (identitas pembuat)

IF2211 Strategi Algoritma – Tugas Besar 7

● Laporan dikumpulkan hari Minggu, 1 Juni 2025 pada alamat Google Form berikut paling lambat pukul 23.59 :

https://bit.ly/4hmMMs0

● Adapun pertanyaan terkait tugas besar ini bisa disampaikan melalui QnA berikut:

https://bit.ly/4iSjlPk

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy adalah pendekatan dalam pemrograman yang memecahkan persoalan optimasi dengan cara yang tampaknya rakus. Pendekatan ini berfokus pada pengambilan keputusan sekarang dengan harapan bahwa setiap langkah akan membawa kita lebih dekat ke solusi akhir yang optimal.

Dalam konteks greedy, kita selalu memilih opsi yang paling menguntungkan saat ini tanpa mempertimbangkan konsekuensi di masa depan.
[https://fikti.umsu.ac.id/algoritma-greedy-pengertian-jenis-dan-contoh-program/]

Untuk mempertimbangkan pengambilan diamond bot ini menggunakan algoritma knapsack greedy by ratio(density) yang mana density = $\frac{Points}{distance}$ algoritma greedy knapsack adalah algoritma untuk menyelesaikan masalah knapsack yang mana terdapat sebuah berat, profit dan densitas sebagai bahan untuk membuat keputusan barang mana yang di ambil

Lalu untuk menentukan rute bot ini menggunakan algoritma dijkstra. Algoritme Dijkstra, (sesuai penemunya Edsger Dijkstra), adalah sebuah algoritma yang dipakai dalam memecahkan permasalahan jarak terpendek (shortest path problem) untuk sebuah graf berarah (directed graph). [https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/]

2.2 Cara Kerja Program

Program bot **LightningYaqueen** dirancang untuk bermain secara efisien dalam permainan Diamonds dengan tujuan mengumpulkan diamond sebanyak-banyaknya sambil menghindari risiko kehilangan diamond akibat tackle lawan. Bot menerima data kondisi papan secara real-time, termasuk posisi bot sendiri, posisi diamond (merah dan biru), posisi lawan, dan status inventory.

Setiap giliran, bot menentukan langkah berikutnya dengan mempertimbangkan beberapa hal:

- Menggunakan algoritma Dijkstra untuk menghitung jarak terpendek dari posisi bot ke seluruh titik pada papan, termasuk diamond dan base.

- Melakukan pemilihan diamond menggunakan metode greedy mirip knapsack dengan memperhatikan nilai diamond dan total jarak tempuh (bot → diamond → base).

- Menyesuaikan prioritas pengambilan diamond merah atau biru sesuai kapasitas inventory yang tersisa.

- Mendeteksi keberadaan lawan di sekitar, dan jika ada lawan yang mengancam bot (berjarak dekat), bot memutuskan apakah harus segera kembali ke base untuk menyimpan diamond atau melakukan tackle balik terlebih dahulu.

- Jika inventory penuh atau hampir penuh dan tidak ada diamond yang layak diambil, bot langsung kembali ke base.

Dengan cara kerja ini, bot mampu bergerak secara adaptif dan strategis sehingga memaksimalkan perolehan poin dan mengurangi risiko kehilangan diamond.

2.2.1 Cara Implementasi Program

Implementasi bot **LightningYaqueen** dilakukan dengan menggunakan bahasa Python dan menerapkan konsep algoritma greedy yang dipadukan dengan heuristik jalur terpendek menggunakan algoritma Dijkstra. Berikut gambaran implementasi secara teknis:

1. **Pembuatan Grid**

   Bot membuat representasi grid kosong berdasarkan ukuran papan permainan yang diterima dari data board.

2. **Pencarian Jalur Terpendek**

   Menggunakan algoritma Dijkstra, bot menghitung jarak terpendek dari posisi saat ini ke seluruh titik pada papan. Informasi ini digunakan untuk menentukan jalur optimal menuju target.

3. **Pemilihan Diamond dengan Pendekatan Knapsack Greedy**

   Bot memilih kombinasi diamond yang optimal berdasarkan rasio nilai diamond dan total jarak tempuh (bot ke diamond ditambah diamond ke base), sambil memperhatikan kapasitas inventory yang terbatas.

4. **Deteksi Lawan dan Mode Taktik**

   Bot memeriksa keberadaan lawan di sekitar dan mengaktifkan strategi bertahan dengan prioritas kembali ke base jika ada ancaman, atau menyerang balik jika kondisi menguntungkan.

5. **Rekonstruksi Jalur dan Penentuan Langkah**

   Jalur ke target yang dipilih direkonstruksi dari hasil Dijkstra, dan bot menentukan langkah satu kotak berikutnya menuju target.

6. **Pengembalian Aksi Gerak**

   Bot mengembalikan arah gerak (delta_x, delta_y) sebagai langkah berikutnya yang akan dieksekusi dalam permainan.

2.2.2 Menjalankan Bot Program

Menjalankan bot **LightningYaqueen** melibatkan beberapa tahapan mulai dari persiapan, registrasi, hingga proses pengambilan keputusan secara real-time selama permainan berlangsung. Berikut penjelasan alur menjalankan bot secara umum:

1. **Persiapan dan Setup**
   Sebelum menjalankan bot, environment perlu dipersiapkan dengan menginstal Python dan dependencies yang dibutuhkan. Bot dijalankan dengan memanggil file utama (**main.py**) dengan argumen yang menentukan logic bot (**--logic LightningYaqueen**), serta data akun seperti email, password, dan nama bot.

2. **Registrasi dan Join Board**
   Saat dijalankan, program akan melakukan registrasi bot ke server game menggunakan email dan password. Jika bot sudah terdaftar, bot akan langsung menggunakan token autentikasi yang ada. Setelah itu, bot akan bergabung ke papan permainan (board) yang telah ditentukan atau memilih board aktif secara otomatis.

3. **Loop Permainan**
   Setelah bergabung, bot memasuki loop utama di mana ia secara terus-menerus menerima data kondisi papan terbaru dari server. Pada setiap giliran, fungsi **next_move** dari class **LightningYaqueen** dipanggil untuk menentukan langkah berikutnya berdasarkan kondisi papan dan status bot saat itu.

4. **Pengiriman Perintah Gerak**
   Langkah yang dihasilkan oleh fungsi **next_move** dikirim ke server sebagai perintah gerak bot. Server akan memproses pergerakan ini dan mengupdate kondisi papan. Bot kemudian menerima update terbaru dan mengulangi proses pengambilan keputusan.

5. **Pengaturan Delay dan Sinkronisasi**
   Program mengatur delay atau jeda waktu antara perintah gerak agar tidak melanggar batas frekuensi komunikasi dengan server, menjaga kestabilan dan kinerja bot selama pertandingan.

6. **Akhir Permainan**
   Proses ini berulang hingga waktu permainan habis atau bot keluar dari papan. Setelah permainan selesai, hasil skor akan ditampilkan sesuai dengan peraturan game.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Proses mapping merupakan tahap penting dalam penerapan algoritma greedy pada permainan Diamonds. Pada tahap ini, elemen-elemen dalam permainan dipetakan ke dalam komponen dasar algoritma greedy sebagai berikut:

- **Himpunan Kandidat (Candidates)**

  Dalam konteks permainan, himpunan kandidat terdiri dari seluruh diamond yang tersedia di papan, baik diamond merah yang bernilai 2 poin maupun diamond biru yang bernilai 1 poin. Selain itu, posisi bot lawan juga menjadi kandidat ketika bot mengaktifkan mode agresif untuk melakukan tackle lawan yang membawa diamond.

- **Himpunan Solusi (Solution Set)**

  Solusi yang mungkin adalah langkah-langkah pergerakan bot untuk mencapai diamond atau lawan yang dipilih sebagai target. Setiap solusi merepresentasikan posisi tujuan selanjutnya dalam papan.

- **Fungsi Seleksi (Selection Function)**

  Fungsi ini bertugas memilih kandidat terbaik berdasarkan kriteria tertentu. Pada bot ini, pemilihan didasarkan pada prioritas nilai diamond dan efisiensi jarak yang dihitung menggunakan algoritma Dijkstra untuk jalur terpendek. Bot memilih diamond yang memberikan nilai maksimum dengan jarak tempuh minimum, serta memilih lawan terdekat saat mode agresif aktif.

- **Fungsi Kelayakan (Feasibility Function)**

  Fungsi kelayakan memastikan bahwa kandidat yang dipilih memenuhi batasan permainan, seperti kapasitas inventory yang terbatas dan validitas langkah dalam batas papan permainan.

Bot menghindari mengambil diamond merah jika inventory hampir penuh untuk mencegah kelebihan kapasitas.

- **Fungsi Objektif (Objective Function)**
  Tujuan utama bot adalah memaksimalkan jumlah poin yang diperoleh dari pengumpulan diamond dan tackle lawan secara efisien. Dengan demikian, fungsi objektif mengarahkan bot untuk mengoptimalkan perolehan poin dengan strategi greedy yang adaptif berdasarkan kondisi permainan saat ini.

Dengan proses mapping ini, setiap langkah bot merupakan pilihan lokal terbaik yang secara konsisten diarahkan untuk mencapai tujuan optimal dalam permainan.

3.2 Eksplorasi Alternatif Solusi Greedy

Strategi greedy yang diterapkan pada bot **LightningYaqueen** saat ini menggunakan pendekatan pemilihan diamond secara lokal berdasarkan rasio nilai dan jarak total (dari posisi bot ke diamond serta dari diamond ke base). Bot juga menggunakan algoritma Dijkstra untuk menentukan jalur terpendek menuju target sehingga langkah yang diambil efisien dan cepat. Pendekatan ini bersifat seleksi lokal, di mana setiap keputusan dibuat berdasarkan kondisi saat ini tanpa mempertimbangkan keseluruhan rencana jangka panjang.

Sebagai alternatif, terdapat pendekatan greedy lain yang dikenal sebagai **Greedy Matching atau Assignment**. Pendekatan ini berfokus pada pencocokan optimal antara dua himpunan, yaitu posisi bot dan posisi diamond yang tersedia di papan. Alih-alih memilih target secara independen di setiap langkah, Greedy Matching mencoba menentukan pasangan bot-diamond yang secara kolektif memberikan hasil terbaik, yaitu memaksimalkan total poin yang bisa diperoleh dengan meminimalkan biaya (jarak atau waktu tempuh).

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Dalam konteks permainan Diamonds, Greedy Matching dapat diterapkan dengan cara membangun matriks biaya yang mengukur jarak antara bot dan setiap diamond, kemudian menggunakan algoritma pencocokan seperti Hungarian Algorithm untuk menemukan pasangan optimal yang meminimalkan total jarak tempuh pengumpulan diamond. Dengan pendekatan ini, bot dapat menyusun urutan pengambilan diamond secara lebih terstruktur dan menghindari perjalanan yang tidak efisien.

Meskipun Greedy Matching memiliki potensi untuk menghasilkan strategi yang lebih optimal secara global, implementasinya cenderung lebih kompleks dan membutuhkan waktu komputasi yang lebih besar. Hal ini dapat menjadi kendala dalam permainan real-time seperti Diamonds, di mana bot harus membuat keputusan dengan cepat berdasarkan data yang terus berubah.

Sebaliknya, strategi yang digunakan saat ini lebih sederhana dan cepat dalam pengambilan keputusan, sehingga lebih cocok untuk aplikasi waktu nyata. Strategi ini juga tetap efektif dengan kombinasi heuristik jalur terpendek dan seleksi lokal yang adaptif terhadap kondisi permainan.

Dengan demikian, meskipun Greedy Matching menawarkan keuntungan dari sisi optimalitas global, strategi greedy seleksi lokal yang diimplementasikan dalam bot **LightningYaqueen** menjadi pilihan yang lebih praktis dan efisien untuk konteks tugas ini.

3.4 Strategi Greedy yang Dipilih

Setelah mempertimbangkan berbagai alternatif strategi greedy, saya memutuskan untuk menggunakan kombinasi algoritma **Greedy Knapsack** dan algoritma **Dijkstra** sebagai pendekatan utama dalam pengembangan bot **LightningYaqueen**. Pemilihan ini didasarkan pada pertimbangan berikut:

1. Kesesuaian dengan Batasan Inventory (Greedy Knapsack)

Permainan Diamonds membatasi kapasitas inventory bot, sehingga bot harus memilih diamond dengan nilai maksimal tanpa melebihi kapasitas tersebut.

Algoritma Greedy Knapsack sangat cocok untuk masalah ini karena secara efisien memilih kombinasi diamond yang memberikan nilai tertinggi berdasarkan rasio nilai terhadap biaya (dalam hal ini jarak tempuh).

Pendekatan ini membantu bot mengoptimalkan pemanfaatan inventory untuk memaksimalkan skor secara lokal pada setiap langkah.

2. Efisiensi Jalur dan Kecepatan Respon (Dijkstra)

Untuk menentukan langkah pergerakan bot menuju target secara efisien, algoritma Dijkstra digunakan sebagai metode pencarian jalur terpendek dalam grid permainan. Algoritma ini memastikan bot dapat menghitung rute optimal menuju diamond atau base dengan mempertimbangkan kondisi papan terkini. Dengan jalur terpendek, bot dapat menghemat waktu pergerakan sehingga dapat mengumpulkan diamond lebih cepat dan mengurangi risiko terjebak atau tertangkap lawan.

3. Keseimbangan antara Kompleksitas dan Performa

Kombinasi kedua algoritma ini memberikan keseimbangan antara kompleksitas perhitungan dan performa bot secara keseluruhan. Greedy Knapsack memberikan metode pemilihan target yang optimal secara lokal dengan waktu komputasi yang ringan, sementara Dijkstra memberikan solusi jalur optimal tanpa membutuhkan eksplorasi yang berlebihan. Hal ini membuat bot mampu beroperasi secara real-time dalam permainan dengan respon yang cepat dan efektif.

4. Adaptabilitas Strategi

Implementasi strategi ini juga memungkinkan integrasi mudah dengan fitur tambahan seperti mode agresif untuk tackle lawan dan mode defensif untuk menghindari risiko kehilangan diamond. Sehingga bot dapat beradaptasi terhadap perubahan situasi dalam permainan secara dinamis.

Dengan pertimbangan tersebut, strategi ini menjadi pilihan terbaik untuk mencapai hasil maksimal dalam kompetisi bot Diamonds.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

```
from typing import Optional, Tuple, List, Dict
from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
import heapq


class LightningYaqueen(BaseLogic):
    def __init__(self):
        self.width = 0
        self.height = 0
        self.grid = []

    def build_grid(self, board: Board):
        self.width = board.width
        self.height = board.height
        self.grid = [[0] * self.width for _ in range(self.height)]

    def neighbors(self, pos: Position) -> List[Position]:
        directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
        result = []
        for dx, dy in directions:
            nx, ny = pos.x + dx, pos.y + dy
            if 0 <= nx < self.width and 0 <= ny < self.height:
                if self.grid[ny][nx] == 0:
                    result.append(Position(ny, nx))
        return result
```

```
    def dijkstra(self, start: Position) -> Tuple[Dict[Tuple[int,
int], int], Dict[Tuple[int, int], Tuple[int, int]]]:
        dist = {(y, x): float('inf') for y in range(self.height) for
x in range(self.width)}
        dist[(start.y, start.x)] = 0
        prev = {}
        heap = [(0, (start.y, start.x))]

        while heap:
            cost, (y, x) = heapq.heappop(heap)
            if cost > dist[(y, x)]:
                continue
            for n in self.neighbors(Position(y, x)):
                nd = (n.y, n.x)
                alt = cost + 1
                if alt < dist[nd]:
                    dist[nd] = alt
                    prev[nd] = (y, x)
                    heapq.heappush(heap, (alt, nd))
        return dist, prev

    def reconstruct_path(self, prev, start: Position, end: Position)
-> List[Position]:
        path = []
        current = (end.y, end.x)
        start_pos = (start.y, start.x)
        while current != start_pos:
            path.append(Position(current[0], current[1]))
            current = prev.get(current)
            if current is None:
                return []
        path.reverse()
        return path

    def select_diamonds_knapsack(self, diamonds: List[GameObject],
dist: Dict[Tuple[int, int], int],
                                 base_pos: Position, inventory_max:
int, exclude_red: bool = False) -> List[GameObject]:
        def total_distance(d: GameObject):
```

```
            return dist.get((d.position.y, d.position.x),
float('inf')) + abs(d.position.x - base_pos.x) + abs(d.position.y -
base_pos.y)

        candidates = []
        for d in diamonds:
            if exclude_red and d.properties.points == 2:
                continue
            dist_total = total_distance(d)
            if dist_total == 0:
                dist_total = 1
            ratio = d.properties.points / dist_total
            candidates.append((ratio, d))

        candidates.sort(key=lambda x: x[0], reverse=True)

        selected = []
        total_items = 0
        for ratio, d in candidates:
            if total_items < inventory_max:
                selected.append(d)
                total_items += 1
            else:
                break

        return selected

    def next_move(self, board_bot: GameObject, board: Board) ->
Tuple[int, int]:
        self.build_grid(board)
        current_pos = board_bot.position
        inventory = board_bot.properties.diamonds or 0
        inventory_max = board_bot.properties.inventory_size or 5
        base_pos = board_bot.properties.base

        dist, prev = self.dijkstra(current_pos)

        diamonds = board.diamonds or []
        enemy_bots = [b for b in (board.bots or []) if
b.properties.name != board_bot.properties.name]
```

```
        # Cek ada musuh di sekitar (jarak 1)
        enemy_nearby = any(dist.get((b.position.y, b.position.x),
float('inf')) == 1 for b in enemy_bots)

        # Pilih diamond merah dan biru terpisah
        red_diamonds = [d for d in diamonds if d.properties.points
== 2]
        blue_diamonds = [d for d in diamonds if d.properties.points
== 1]

        # Hitung jarak ke base
        dist_to_base = dist.get((base_pos.y, base_pos.x),
float('inf'))

        # Tentukan target:
        # Jika inventory >=4, abaikan diamond merah dan cari biru
        if inventory >= 4:
            candidate_diamonds =
self.select_diamonds_knapsack(blue_diamonds, dist, base_pos,
inventory_max - inventory)
        else:
            # Gabung diamond merah + biru untuk knapsack
            candidate_diamonds =
self.select_diamonds_knapsack(diamonds, dist, base_pos,
inventory_max - inventory)

        # Jika ada musuh dekat dan inventory >0 dan jarak base lebih
dekat dari diamond terdekat
        if enemy_nearby and inventory > 0:
            # Cari jarak diamond terdekat
            if candidate_diamonds:
                candidate_diamonds.sort(key=lambda d:
dist.get((d.position.y, d.position.x), float('inf')))
                dist_to_nearest_diamond =
dist.get((candidate_diamonds[0].position.y,
candidate_diamonds[0].position.x), float('inf'))
            else:
                dist_to_nearest_diamond = float('inf')
```

```
            if dist_to_base < dist_to_nearest_diamond:
                target = base_pos
            else:
                target = candidate_diamonds[0].position if
candidate_diamonds else base_pos
        else:
            # Jika inventory penuh atau hampir penuh dan tidak ada
diamond, langsung ke base
            if inventory >= inventory_max or (inventory ==
inventory_max -1 and not candidate_diamonds):
                target = base_pos
            else:
                target = candidate_diamonds[0].position if
candidate_diamonds else base_pos

        path = self.reconstruct_path(prev, current_pos, target)
        if not path:
            return 0, 0

        next_step = path[0]
        dx = next_step.x - current_pos.x
        dy = next_step.y - current_pos.y

        if abs(dx) + abs(dy) != 1:
            return 0, 0
        return dx, dy
```

4.1.2 Penjelasan Alur Program

1. Membangun Representasi Grid Papan

Program membuat grid dua dimensi (self.grid) sesuai ukuran papan (width x height). Grid merepresentasikan posisi yang dapat dilalui (nilai 0) dan halangan.

2. Menghitung Jalur Terpendek dengan Algoritma Dijkstra

Menghitung jarak terpendek dari posisi bot saat ini ke seluruh posisi lain di papan. Menghasilkan dictionary dist (jarak minimum ke setiap titik) dan prev (jejak jalur) untuk rekonstruksi path.

3. Identifikasi Objek dan Kondisi

Mengambil data posisi bot, inventory diamond yang dimiliki, kapasitas inventory maksimal, dan posisi base. Mengumpulkan daftar diamond (merah dan biru) dan bot musuh.

4. Pemilihan Target Diamond dengan Pendekatan Greedy by Density

Menghitung rasio poin diamond terhadap total jarak tempuh (bot → diamond → base). Memilih diamond dengan rasio tertinggi sesuai kapasitas inventory yang tersisa. Jika inventory sudah hampir penuh, diamond merah diabaikan untuk menghindari kelebihan kapasitas.

5. Pengambilan Keputusan Strategis Berdasarkan Ancaman Musuh

Jika ada musuh sangat dekat (jarak 1) dan bot membawa diamond, bot mempertimbangkan untuk kembali ke base demi keamanan. Memilih target yang paling aman antara diamond terdekat atau base.

6. Rekonstruksi Jalur dan Penentuan Langkah Berikutnya

Menggunakan data prev untuk merekonstruksi jalur terpendek menuju target yang dipilih. Mengambil langkah pertama dari jalur tersebut sebagai gerakan saat ini.

7. Pelaksanaan Langkah

Mengembalikan perbedaan koordinat (dx, dy) untuk langkah satu kotak ke arah target.

4.2 Struktur Data yang Digunakan

1. Grid Dua Dimensi (self.grid)

Tipe: List of Lists (List[List[int]])

Deskripsi: Matriks dua dimensi yang merepresentasikan papan permainan secara grid, dengan ukuran height x width.

Fungsi: Nilai 0 menunjukkan posisi yang dapat dilalui bot (jalan). Posisi lain dapat diisi dengan nilai berbeda (misalnya halangan). Digunakan untuk pengecekan validitas langkah dan penentuan tetangga dalam algoritma jalur terpendek.

2. Objek Posisi (Position)

Tipe: Class dengan atribut koordinat (x: int, y: int)

Deskripsi: Menyimpan posisi suatu titik di papan permainan.

Fungsi: Digunakan sebagai parameter dalam fungsi navigasi, identifikasi lokasi bot, diamond, base, dan musuh.

3. Objek Game (GameObject)

Tipe: Class kompleks dengan properti seperti:

Position: Position — posisi objek properties — atribut khusus seperti jumlah diamond, nama, nilai diamond, base, dll.

Fungsi: Merepresentasikan entitas di dalam game seperti bot, diamond, dan musuh.

4. Struktur Data Jalur dan Jarak

Dist

Tipe: Dictionary dengan key (y, x) dan value int (jarak)

Fungsi: Menyimpan jarak terpendek dari posisi awal ke setiap titik pada grid hasil algoritma Dijkstra.

Prev

Tipe: Dictionary dengan key (y, x) dan value (y, x) (posisi sebelumnya)

Fungsi: Menyimpan informasi jalur agar bisa merekonstruksi rute terpendek dari posisi akhir ke awal.

5. List Objek

List Diamond: Menyimpan semua objek diamond (List[GameObject]) yang ada di papan. Dipisah lagi menjadi red_diamonds dan blue_diamonds berdasarkan nilai poin.

List Bot Musuh: Menyimpan semua bot selain bot utama (List[GameObject]) untuk pertimbangan musuh dalam pengambilan keputusan.

6. Heap/Priority Queue

Digunakan dalam implementasi algoritma Dijkstra untuk memilih node dengan jarak terkecil berikutnya secara efisien.

4.3 Pengujian Program

4.3.1 Skenario Pengujian

No	Skenario	Hasil yang diharapkan
1	Diamond berada dekat dan tidak terhalang rintangan	Bot Bergerak langsung ke diamond
2	Diamond berada lebih dari satu dan bernilai beda beda	Bot menentukan diamond yang akan diambil dengan greedy by ratio(density)
3	Ada bot lain menghalangi jalur	Menentukan rute menggunakan dijkstra
4	Tas penuh atau tidak muat lagi menampung diamond	Kembali ke base
5	Diamond bernilai besar tapu jauh	Bot menentukan apakah diamond tersebut adalah yang memiliki ratio tertinggi atau tidak

4.3.2 Hasil Pengujian dan Analisis

No	Hasil bot	Sesuai/Tidak Sesuai
1	Bot bergerak langsung ke arah diamond	sesuai
2	Bot memilih diamond dengan density atau ratio yang terbaik	sesuai
3	Bot menghindari bot lain namun ada suatu kondisi dimana bot akan mencoba untuk men-tacle bot lain, dan tetap mencapai diamond	sesuai
4	Bot Langsung kembali ke base	sesuai
5	Bot akan tetap mempertimbangkan nilai ratio dari diamond	sesuai

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Implementasi LightningYaqueen merupakan aplikasi algoritma greedy by ratio(density) dan Algoritma Dijkstra yang memilih solusi lokal terbaik (diamond/lawan terdekat dengan jarak optimal) pada tiap langkah dengan tujuan optimasi skor secara keseluruhan.

5.2 Saran

Masih banyak hal yang bisa dikembangkan dalam bot ini, seperti Tambahkan mekanisme untuk menilai risiko, seperti memutuskan kapan harus meninggalkan diamond dan lari saat dikejar banyak musuh, dan Pertimbangkan prioritas diamond berdasarkan situasi permainan, misalnya diamond merah lebih penting saat musuh jauh, diamond biru saat musuh dekat.

LAMPIRAN

A. Repository Github (link)

DAFTAR PUSTAKA

Format IEEE ya