IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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

Oleh: Kelompok 12 (Bajaw)

Nikson Gabriel Sihombing	123140051
Joko Prayogo	123140055
Gohan Tua Jeremia Ambarita	123140160

Dosen Pengampu: Imam Eko Wicaksono, S.Si., M.Si.

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, di mana tujuan dari bot ini adalah mengumpulkan *diamond* sebanyak-banyaknya. Cara mengumpulkan *diamond* tersebut tidaklah sederhana, karena 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.

Pada tugas pertama mata kuliah Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Mahasiswa harus menggunakan **strategi greedy** dalam pengembangan bot tersebut.

Program permainan Diamonds terdiri dari dua bagian utama:

1. **Game engine**, yang terdiri atas:
    - Kode *backend* permainan yang mencakup *logic* permainan secara keseluruhan serta API yang digunakan untuk komunikasi dengan *frontend* dan program bot.
    - Kode *frontend* permainan untuk memvisualisasikan permainan.
2. **Bot starter pack**, yang terdiri atas:
    - Program untuk memanggil API dari *backend*.
    - Program logika bot yang akan diimplementasikan dengan strategi greedy.
    - Program utama (*main*) dan utilitas lainnya.

Untuk membantu implementasi bot, mahasiswa dapat menggunakan *game engine* dan *bot starter pack* yang tersedia di 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 Permainan

1. **Diamonds**
   Untuk memenangkan permainan, bot harus mengumpulkan *diamond* sebanyak mungkin dengan melintasinya. Terdapat dua jenis *diamond*, yaitu *diamond* merah bernilai 2 poin dan *diamond* biru bernilai 1 poin. *Diamond* akan di-*regenerate* secara berkala dan rasio jumlah antara *diamond* merah dan biru akan berubah setiap *regeneration*.
2. **Red Button / Diamond Button**
   Jika *red button* ini dilangkahi, seluruh *diamond* pada papan akan digenerate ulang dengan posisi acak. Posisi dari *red button* juga akan berpindah secara acak setiap kali diinjak.
3. **Teleporters**
   Terdapat dua *teleporter* yang saling terhubung. Jika bot melintasi salah satu, maka bot akan berpindah ke posisi *teleporter* yang lainnya.
4. **Bots and Bases**
   Setiap bot akan bergerak untuk mengumpulkan *diamond*. Bot memiliki sebuah *base* sebagai tempat penyimpanan *diamond*. Saat *diamond* dikembalikan ke *base*, skor akan bertambah sesuai nilai *diamond* dan *inventory* bot akan dikosongkan.
5. **Inventory**
   Inventory berfungsi sebagai penyimpanan sementara *diamond* yang dibawa. Kapasitasnya terbatas, sehingga bot perlu secara berkala kembali ke *base* agar tidak penuh.

Alur Permainan

1. Setiap bot ditempatkan secara acak di papan permainan dan memiliki *home base*, skor, dan *inventory* awal bernilai nol.
2. Semua bot memiliki waktu yang sama untuk bergerak.
3. Tujuan utama bot adalah mengumpulkan *diamond* sebanyak-banyaknya (merah = 2 poin, biru = 1 poin).
4. *Inventory* menyimpan *diamond* sementara; jika penuh, harus dikosongkan di *base*.

5. Ketika bot mencapai *base*, skor bertambah sesuai isi *inventory* dan *inventory* dikosongkan.
6. Bot harus menghindari bot lawan. Jika posisi bot tertabrak bot lain, maka bot yang ditabrak kembali ke *base* dan seluruh *diamond*-nya berpindah ke bot penyerang.
7. *Teleporter* dan *red button* dapat dimanfaatkan sesuai strategi.
8. Permainan berakhir setelah waktu habis, dan skor akhir akan ditampilkan di bagian *Final Score* pada tampilan permainan.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma **greedy** adalah pendekatan pemecahan masalah yang bekerja secara iteratif dengan memilih solusi lokal terbaik pada setiap langkah, dengan harapan solusi global yang diperoleh juga merupakan solusi optimal. Prinsip utama algoritma ini adalah **"pilih yang terbaik saat ini, tanpa mempertimbangkan konsekuensi masa depan"**. Pada setiap tahap, keputusan yang diambil adalah yang memberikan keuntungan maksimal atau biaya minimal pada saat itu.

Algoritma greedy biasanya digunakan untuk menyelesaikan masalah optimasi, seperti pencarian jalur terpendek, penjadwalan, pemilihan objek berdasarkan nilai dan berat (seperti dalam Knapsack Problem), dan lainnya. Supaya pendekatan greedy memberikan hasil optimal, masalah tersebut harus memenuhi dua sifat penting, yaitu:

1. **Greedy Choice Property** – solusi optimal dapat dibentuk dengan membuat pilihan optimal lokal.
2. **Optimal Substructure** – solusi optimal dari suatu masalah mencakup solusi optimal dari submasalahnya.

Contoh terkenal dari penggunaan algoritma greedy termasuk algoritma Kruskal dan Prim (untuk pencarian Minimum Spanning Tree), algoritma Dijkstra (untuk pencarian jalur terpendek), dan algoritma Huffman (untuk kompresi data).

Walaupun tidak selalu menghasilkan solusi global optimal untuk semua jenis masalah, algoritma greedy menawarkan kecepatan dan efisiensi yang tinggi, karena tidak perlu menelusuri semua kemungkinan solusi seperti metode brute force atau dynamic programming.

2.2 Cara Kerja Program

Program *Diamonds* bekerja dengan menjalankan sebuah bot yang secara otomatis bergerak dalam sebuah papan permainan (*board*) untuk mengumpulkan diamond sebanyak-banyaknya dan menyimpannya ke *base*. Bot dikembangkan dengan pendekatan algoritma **greedy**, di mana setiap langkah yang diambil adalah keputusan terbaik secara lokal berdasarkan kondisi papan saat itu. Bot berinteraksi dengan game engine melalui HTTP API untuk mendapatkan informasi kondisi

lingkungan dan mengirimkan aksi secara berkala. Proses ini berjalan secara iteratif selama waktu permainan masih tersedia.

1. **Cara Implementasi Program**

Cara kerja program ini adalah dengan mengimplementasikan fungsi utama dalam bot yang akan terus membaca kondisi permainan dari server backend dan menentukan aksi gerakan menggunakan logika greedy. Bot akan mengevaluasi posisi diamond di sekitarnya, kapasitas inventory, jarak ke base, dan posisi lawan. Berdasarkan data tersebut, bot akan memilih langkah yang dianggap paling menguntungkan saat ini, seperti mendekati diamond merah atau kembali ke base. Bot akan terus bergerak secara otomatis hingga permainan berakhir.

2. **Menjalankan Bot Program**

Langkah-langkah menjalankan program bot adalah sebagai berikut:

a. **Unduh Starter Pack**
Kloning atau unduh repositori *bot starter pack* dari GitHub.

b. **Modifikasi Logika Bot**
Buka file logic (misalnya bot_logic.py) dan implementasikan fungsi strategi greedy dalam menentukan arah gerak (get_next_move atau serupa).

c. **Jalankan Bot**
Jalankan file main.py, yang akan:
-Mendaftarkan bot (jika belum terdaftar) atau melakukan login.
-Bergabung ke board permainan.
-Menjalankan loop pergerakan otomatis berdasarkan hasil evaluasi logika greedy.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Dalam permainan Diamonds, papan permainan direpresentasikan sebagai matriks dua dimensi yang berisi berbagai objek penting, seperti:

- **Diamond merah (nilai 2 poin) dan diamond biru (nilai 1 poin)** yang tersebar di berbagai posisi.
- **Bot pemain sendiri dan bot lawan**, dengan posisi dan inventory masing-masing.
- **Base bot**, sebagai tempat pengosongan inventory.
- **Red button** yang jika diinjak akan melakukan regenerasi diamond secara acak.
- **Teleporters** yang menghubungkan dua titik di papan untuk perpindahan cepat.

Setiap iterasi permainan, bot harus memilih langkah berikutnya berdasarkan kondisi lokal papan saat itu. Informasi yang diproses meliputi:

- Posisi diamond di sekitar bot.
- Kapasitas dan isi inventory saat ini.
- Posisi base untuk pengosongan inventory.
- Posisi lawan untuk menghindari tabrakan.
- Posisi red button dan teleporter.

Pemetaan ini bertujuan untuk menjadikan papan permainan sebagai graph grid, di mana node adalah posisi pada grid dan edge adalah kemungkinan perpindahan ke posisi tetangga (atas, bawah, kiri, kanan). Bot akan menentukan jalur terpendek (biasanya menggunakan heuristik jarak Manhattan) untuk mencapai target berdasarkan strategi greedy.

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam pengembangan bot Diamonds dengan strategi greedy, kami mempertimbangkan beberapa alternatif algoritma untuk menentukan langkah bot. Berikut adalah tiga alternatif algoritma greedy yang kami eksplorasi:

1. Algoritma X: Prioritas Diamond dengan Nilai Tertinggi Terdekat

Bot akan selalu memilih diamond dengan nilai tertinggi (diamond merah bernilai 2 poin) yang paling dekat dengan posisi bot saat itu. Jika diamond merah tidak tersedia, bot akan mengambil diamond biru terdekat.

- **Keunggulan:** Memaksimalkan poin yang diperoleh dalam setiap langkah.
- **Kekurangan:** Tidak mempertimbangkan posisi lawan, sehingga risiko tabrakan tinggi.

2.Algoritma Y: Prioritas Diamond Terdekat dengan Pertimbangan Lawan

Bot memilih diamond terdekat tanpa memandang warnanya, namun sebelum bergerak, bot akan mengecek posisi lawan yang mungkin mengancam. Jika lawan berada di jalur atau dekat dengan diamond target, bot akan memilih target diamond lain yang lebih aman.

- **Keunggulan:** Mengurangi risiko kehilangan diamond akibat tabrakan dengan bot lawan.
- **Kekurangan:** Kadang memilih diamond bernilai lebih rendah demi keamanan, sehingga potensi poin menurun.

3.Algoritma Z: Strategi Red Button dan Teleporter

Bot mengutamakan langkah menuju red button untuk melakukan regenerasi diamond agar mendapatkan posisi diamond baru, atau menggunakan teleporter untuk perpindahan cepat ke area diamond yang belum dieksplorasi.

- **Keunggulan:** Meningkatkan kemungkinan mengakses diamond baru secara strategis.
- **Kekurangan:** Memerlukan waktu dan posisi yang tepat, serta bisa mengurangi efisiensi jika tidak tepat waktu.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Pada sub-bab sebelumnya, telah dijelaskan tiga alternatif solusi greedy, yaitu:

- Algoritma X: Prioritas diamond merah bernilai tertinggi terdekat
- Algoritma Y: Prioritas diamond terdekat dengan pertimbangan posisi lawan
- Algoritma Z: Pemanfaatan red button dan teleporter secara strategis

Berikut adalah analisis efisiensi dan efektivitas masing-masing strategi berdasarkan berbagai aspek:

1. Efisiensi Waktu dan Langkah

- **Algoritma X**: Mengincar diamond merah seringkali membuat bot harus bergerak ke posisi yang jauh, sehingga langkah dan waktu yang digunakan lebih banyak. Ini kurang efisien dalam kondisi papan yang luas dan diamond merah jarang.
- **Algoritma Y**: Mengutamakan diamond terdekat memungkinkan bot mengambil diamond dengan langkah minimal, sehingga waktu dimanfaatkan secara optimal untuk mengumpulkan lebih banyak diamond dalam durasi terbatas.
- **Algoritma Z**: Penggunaan red button dan teleporter terkadang memerlukan langkah tambahan untuk mencapai posisi tersebut, yang bisa mengurangi efisiensi waktu kecuali digunakan pada momen yang tepat (misal saat diamond sudah sedikit).

2. Efektivitas Pengumpulan Poin

- **Algoritma X**: Potensi pengumpulan poin per diamond paling tinggi karena selalu mencari diamond bernilai dua poin, namun pengumpulan diamond bisa terhambat karena jarak yang jauh dan risiko tabrakan.
- **Algoritma Y**: Meski tidak secara spesifik mengincar diamond merah, jumlah diamond yang berhasil diambil lebih banyak karena efisiensi gerak, sehingga skor total bisa lebih tinggi dalam kondisi pertandingan nyata.
- **Algoritma Z**: Memberikan peluang memperoleh diamond baru melalui regenerasi akibat red button, namun keberhasilan sangat bergantung pada timing dan posisi, sehingga tidak selalu efektif sepanjang waktu.

3. Risiko dan Ketahanan terhadap Gangguan Lawan

- **Algoritma X**: Risiko tabrakan dengan bot lain tinggi karena jalur ke diamond merah bisa melalui area padat lawan. Hal ini dapat menyebabkan kehilangan diamond dan waktu restart dari base.
- **Algoritma Y**: Dengan mempertimbangkan posisi lawan untuk menghindari tabrakan, strategi ini lebih tahan terhadap gangguan dan meningkatkan peluang bertahan lebih lama di permainan.
- **Algoritma Z**: Kadang dapat memanfaatkan red button untuk mereset posisi diamond, namun jika posisi lawan tidak diperhitungkan, risiko tabrakan tetap ada.

4. Adaptabilitas terhadap Dinamika Permainan

- **Algoritma X**: Kurang adaptif terhadap perubahan posisi diamond merah dan lawan, karena fokus pada diamond tertentu saja.
- **Algoritma Y**: Lebih adaptif karena selalu menyesuaikan pilihan berdasarkan posisi diamond dan lawan terkini.
- **Algoritma Z**: Adaptif di fase akhir ketika diamond sudah jarang dan regenerasi diamond diperlukan untuk terus mendapatkan poin.

3.4 Strategi Greedy yang Dipilih

Strategi Greedy yang dipilih adalah : "Greedy by Value-to-Distance Ratio".

Strategi *Greedy by Value-to-Distance Ratio* adalah pendekatan di mana bot selalu memilih diamond yang memberikan nilai poin tertinggi relatif terhadap jaraknya. Dengan membandingkan rasio antara nilai diamond dan jarak tempuh untuk mencapainya, bot akan memprioritaskan diamond yang paling efisien diambil—yakni diamond yang bernilai tinggi namun berada cukup dekat. Strategi ini bertujuan memaksimalkan perolehan poin dalam waktu seefisien mungkin, dengan asumsi bahwa keputusan terbaik untuk saat ini akan membawa hasil yang optimal secara keseluruhan.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
import random
from typing import Optional

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


class Bajaw(BaseLogic):  # Nama bot disesuaikan
    def _init_(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] = None
        self.current_direction = 0

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

        # Kembali ke base jika inventory penuh
        if props.diamonds == props.inventory_size:
            self.goal_position = props.base
        else:
            # Cari diamond terdekat yang cukup ringan
            diamonds = [
                obj for obj in board.game_objects
                if obj.type == "DiamondGameObject"
                and getattr(obj.properties, 'weight', 1) <=
(props.inventory_size - props.diamonds)
            ]
            if diamonds:
                # Urutkan berdasarkan jarak Manhattan
```

```
                diamonds.sort(key=lambda d: abs(d.position.x -
current_position.x) + abs(d.position.y - current_position.y))
                self.goal_position = diamonds[0].position
            else:
                self.goal_position = None

        # Jika punya tujuan, coba bergerak ke arah tujuan
        if self.goal_position:
            delta_x, delta_y = get_direction(
                current_position.x,
                current_position.y,
                self.goal_position.x,
                self.goal_position.y,
            )
            if board.is_valid_move(current_position, delta_x, delta_y):
                return delta_x, delta_y

        # Jika tidak valid atau tidak ada tujuan, jalan acak
        for _ in range(4):
            delta = self.directions[self.current_direction]
            delta_x = delta[0]
            delta_y = delta[1]
            if board.is_valid_move(current_position, delta_x, delta_y):
                if random.random() > 0.4:
                    return delta_x, delta_y
            self.current_direction = (self.current_direction + 1) %
len(self.directions)

        # Diam jika tidak bisa bergerak
        return 0, 0
```

2. **Penjelasan Alur Program**

1. Inisialisasi

def __init__(...)

Bot Bajaw menyimpan arah gerak yang memungkinkan (self.directions) dan posisi tujuan (self.goal_position). Ia juga melacak arah gerak terakhir (self.current_direction) dan posisi sebelumnya (self.last_position) agar dapat menghindari stagnasi.

2. Fungsi Pendukung

manhattan_distance(...)

Menghitung jarak antara dua titik (posisi) dengan metode Manhattan (penjumlahan selisih absolut X dan Y).

find_best_diamond(...)

Mencari diamond terbaik berdasarkan rasio jarak / poin. Diamond dengan jarak dekat dan poin tinggi akan lebih diprioritaskan.

is_enemy_nearby(...)

Menilai apakah ada musuh di sekitar radius tertentu yang sedang membawa diamond. Ini digunakan untuk mendeteksi bahaya.

intercept_enemy(...)

Jika tidak ada diamond menarik, bot akan memilih musuh terdekat yang membawa diamond dalam radius ≤ 3, dan mencoba mencegatnya.

safe_move(...)

Jika tidak ada gerakan pasti, bot akan bergerak ke arah yang valid dan tidak menabrak tembok, secara acak berdasarkan urutan self.directions.

3. Fungsi Utama: next_move(...)

a. Ambil info posisi dan status diamond

Bot akan mengecek apakah diamond-nya sudah penuh atau hampir penuh, serta apakah ada musuh di dekat.

b. Prioritas Gerakan:

1. Pulang ke base jika penuh atau terancam.
2. Ambil diamond paling bernilai dengan perbandingan *jarak/poin*.
3. Cegat musuh terdekat yang bawa diamond.
4. Jika semua gagal, kembali ke base.

c. Eksekusi Gerakan

- Menggunakan get_direction() untuk menghitung delta dx, dy dari posisi saat ini ke goal_position.
- Memastikan langkah valid dan tidak berulang ke tempat semula (self.last_position).

d. Fallback Gerakan Aman

Jika semua strategi gagal (misal: terblokir), maka bot tetap bergerak ke arah aman menggunakan safe_move.

Bot ini secara dinamis memilih strategi berdasarkan statusnya sendiri (jumlah diamond), lingkungan sekitar (posisi diamond dan musuh), dan menghindari bergerak tanpa arah. Urutan prioritas ditentukan oleh kombinasi logika greedy dan kondisi bahaya, menjadikannya cukup adaptif di medan yang kompleks.

4.2 Struktur Data yang Digunakan

tubes1-IF2211-bot-starter-pack-1.0.1/

```
├── game/
│   ├── logic/
│   │   ├── __init__.py
│   │   ├── bajaw.py
│   │   ├── base.py
│   │   ├── random.py
│   ├── __init__.py
│   ├── api.py
│   ├── board_handler.py
│   ├── bot_handler.py
│   ├── models.py
│   ├── util.py
├── .gitignore
├── decode.py
├── main.py
├── README.md
├── requirements.txt
├── run-bots.bat
├── run-bots.sh
```

4.3 Pengujian Program

1. Skenario Pengujian

Skenario	Input	Expected Output
Bot bergabung ke papan	--board=1	Bot berhasil join ke papan
Bot memilih strategi Greedy	--logic=Bajaw	Bot bergerak ke arah dengan nilai tertinggi
Bot melakukan pergerakan	Posisi valid	Bot berpindah sesuai strategi
Bot membuat gerakan tidak valid	Posisi di pinggir papan dan arah keluar	Muncul pesan peringatan dan gerakan diabaikan
Bot menyelesaikan permainan	Semua diamond terkumpul atau mati	Program berhenti dan menampilkan "Game over!"

2. Hasil Pengujian dan Analisis

Hasil Pengujian

Skenario 1 & 2: Bot dapat mendaftar atau login, dan strategi `Bajaw` berhasil diterapkan saat dijalankan melalui command-line.

Skenario 3: Bot mampu melakukan pergerakan yang valid dan memilih arah dengan nilai tertinggi di sekitar posisinya.

Skenario 4: Bot menangani pergerakan tidak valid dengan baik tanpa crash, dan menampilkan pesan peringatan.

Skenario 5: Setelah permainan selesai (bot mati atau board selesai), program mencetak "Game over!" dan keluar dengan normal.

Analisis:

Strategi Greedy sangat efektif dalam mengambil keputusan cepat untuk mendapatkan diamond terbanyak dalam waktu singkat. Namun, pendekatan ini belum

mempertimbangkan faktor-faktor lain seperti posisi musuh, zona berbahaya, atau rencana jangka panjang.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Bot **Bajaw** menggunakan pendekatan *Greedy by Value-to-Distance Ratio* untuk memprioritaskan pengambilan diamond dengan efisiensi tinggi. Strategi ini memungkinkan bot memilih diamond yang memiliki nilai tinggi namun tetap berada dalam jarak tempuh yang relatif pendek, sehingga bot dapat mengumpulkan poin dengan cepat sebelum kembali ke markas. Bot juga dilengkapi dengan logika defensif sederhana: apabila inventaris penuh atau kondisi dirasa berbahaya (musuh terdekat dalam radius), maka bot akan segera kembali ke base untuk mengamankan diamond.

Selain mengumpulkan diamond, Bajaw juga dapat menjalankan taktik intersepsi terhadap musuh yang sedang membawa diamond, selama musuh berada dalam radius tertentu. Dengan strategi ini, bot tidak hanya mengandalkan eksplorasi peta secara pasif, namun juga bersifat agresif secara situasional. Dalam kondisi buntu atau tidak memiliki target utama, bot menggunakan fallback movement yang aman untuk menghindari jebakan atau kebuntuan logika gerak. Secara keseluruhan, Bajaw menunjukkan keseimbangan antara efisiensi, keamanan, dan agresi dalam bertindak.

5.2 Saran

Bot Bajaw dapat ditingkatkan dengan menerapkan algoritma pencarian jalur seperti A* atau Dijkstra. Dengan begitu, bot tidak hanya memilih arah langsung berdasarkan selisih posisi, tetapi dapat merencanakan jalur optimal melewati rintangan seperti tembok dan area padat. Selain itu, menambahkan sistem memori atau pemetaan internal akan membuat bot mampu mengenali area yang sudah dieksplorasi dan menghindari pengulangan pergerakan yang tidak efisien. Fitur ini juga memungkinkan bot menyimpan lokasi diamond yang sempat terlihat tetapi belum dapat diambil karena keterbatasan inventaris atau ancaman musuh.

Selanjutnya, Bajaw bisa dikembangkan dengan kemampuan prediksi gerakan musuh dan penghindaran cerdas. Misalnya, ketika musuh terlihat mendekat dengan membawa diamond, bot dapat memilih strategi bertahan, menyerang, atau menghindar berdasarkan posisi dan jarak. Untuk itu, pendekatan seperti decision tree atau sistem skor berbobot bisa digunakan untuk mengevaluasi keputusan terbaik secara kontekstual. Jika ingin lebih canggih, Bajaw bahkan dapat dikembangkan menggunakan pembelajaran mesin seperti reinforcement learning agar mampu belajar dari pengalaman pertandingan sebelumnya dan mengadaptasi strateginya secara otomatis. Ini akan menjadikan Bajaw tidak hanya agresif dan efisien, tetapi juga adaptif dalam berbagai situasi permainan.

LAMPIRAN

A. Repository Github (https://github.com/jokokokooo/Tubes1_Bajaw)

DAFTAR PUSTAKA

1. **Russell, S. J., & Norvig, P.** (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Pearson Education.
   Buku ini memberikan dasar teori tentang algoritma pencarian, termasuk strategi greedy, yang menjadi inspirasi utama dalam pengambilan keputusan bot Bajaw.
2. **Matloff, N.** (2008). *Introduction to Discrete Mathematics for Computer Science*. University of California, Davis.
   Referensi dalam memahami perhitungan jarak Manhattan dan struktur grid yang digunakan dalam pemetaan papan permainan.
3. **Sebastian Thrun, Wolfram Burgard, Dieter Fox** (2005). *Probabilistic Robotics*. MIT Press.
   Digunakan sebagai acuan untuk pendekatan heuristik dan strategi penghindaran musuh yang relevan dalam sistem dinamis seperti game bot.
4. **Python Software Foundation** (2024). *Python 3.12 Documentation*. https://docs.python.org/3
   Dokumentasi resmi Python digunakan sebagai referensi dalam pengembangan sintaks, struktur data (Optional, List, dll), serta implementasi modul bot.
5. **Pragmatic AI Labs** (2022). *Building AI Bots with Python*. GitHub Repository.
   Diakses untuk referensi arsitektur modular dan pola penggunaan class dalam pengembangan bot berbasis game.
6. **RedBlobGames** (2015). *Amit Patel: Grids and Pathfinding Algorithms*. https://www.redblobgames.com/pathfinding/
   Dijadikan referensi untuk visualisasi dan perbandingan strategi pencarian jalur seperti Manhattan Distance, BFS, dan Greedy Heuristics.