IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMONDS

Tugas Besar

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

Oleh: Kelompok Algoventure

Riyan Sandi Prayoga	123140176
Tri Putri Sormin	123140013

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

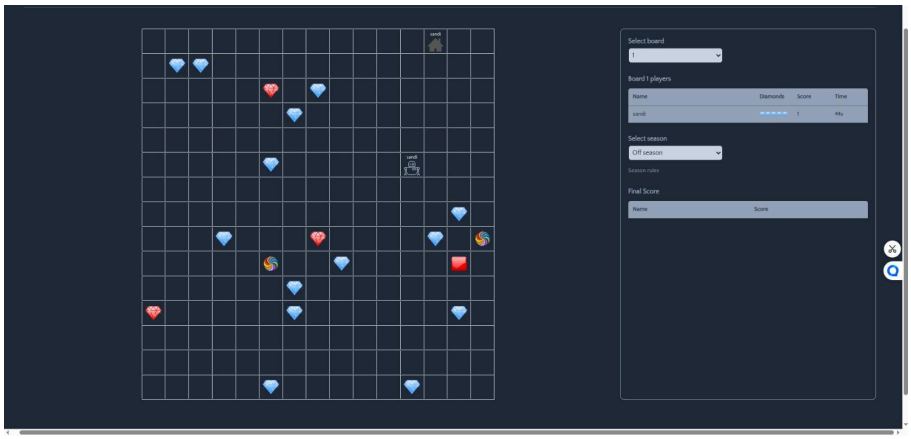
PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2025

DAFTAR ISI

BAB I

DESKRIPSI TUGAS

Tugas besar ini mengharuskan mahasiswa untuk membuat sebuah program bot otomatis menggunakan bahasa pemrograman Python yang mengimplementasikan algoritma Greedy pada permainan strategi berbasis web bernama Diamonds. Permainan ini menuntut peserta untuk merancang bot yang mampu memenangkan pertandingan dengan mengumpulkan diamond sebanyak-banyaknya dan menghindari agar diamond tersebut tidak diambil oleh bot lawan.



Gambar 1.Ilustrasi Permain Diomonds

Bot akan bertanding melawan bot dari kelompok lain dalam sebuah arena permainan digital sebagai bagian dari kompetisi resmi Tubes 1. Oleh karena itu, bot harus menerapkan strategi greedy yang optimal dan efisien agar mampu memaksimalkan perolehan diamond yang memiliki nilai berbeda berdasarkan warna, di mana diamond merah bernilai lebih tinggi dibandingkan diamond biru. Bot juga harus memperhitungkan keterbatasan kapasitas penyimpanan (inventory)

dalam membawa diamond, melakukan penyetoran diamond di base untuk memperoleh skor secara efektif, memanfaatkan teleporter untuk berpindah posisi dengan cepat, serta mengantisipasi keberadaan red button yang dapat mereset posisi diamond di papan permainan.

Selain itu, bot harus mampu menghindari kontak langsung dengan bot lawan karena tabrakan dapat menyebabkan hilangnya seluruh diamond yang sedang dibawa. Strategi greedy yang dirancang harus dikaitkan secara jelas dengan fungsi objektif permainan, yaitu memenangkan pertandingan dengan memperoleh skor tertinggi melalui pengumpulan diamond sebanyak-banyaknya dan meminimalkan risiko kehilangan diamond akibat persaingan dengan bot lain.

Dalam strategi tersebut, setiap komponen dasar algoritma greedy harus dijelaskan secara eksplisit, meliputi himpunan kandidat (candidate set) sebagai pilihan diamond atau aksi yang dapat dipilih bot pada setiap langkah, fungsi seleksi (selection function) yang menentukan aksi terbaik berdasarkan kriteria greedy, fungsi kelayakan (feasibility function) untuk memastikan aksi yang diambil tidak melanggar aturan permainan atau batasan kapasitas, serta fungsi objektif (objective function) yang memaksimalkan skor dan meminimalkan risiko kehilangan diamond.

Tugas ini dikerjakan secara berkelompok dengan anggota minimal dua orang dan maksimal tiga orang, di mana kolaborasi antar kelas maupun antar kampus diperbolehkan. Setiap kelompok wajib mendokumentasikan strategi greedy yang digunakan secara terperinci dalam laporan. Dokumentasi ini akan diperiksa dan diverifikasi secara langsung pada sesi demonstrasi untuk memastikan kesesuaian antara strategi yang ditulis dengan implementasi program bot.

Kreativitas dan inovasi dalam menyusun strategi greedy sangat dianjurkan untuk meningkatkan peluang kemenangan dalam kompetisi antar kelompok. Melalui tugas ini, mahasiswa diharapkan dapat menerapkan teori algoritma greedy dalam konteks nyata, merancang solusi strategis yang berbasis logika dan efektif, mengasah kemampuan kerja sama dalam tim, serta melatih kemampuan pemecahan masalah dalam lingkungan yang kompetitif dan dinamis.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy merupakan salah satu pendekatan dalam pemecahan masalah optimasi di bidang algoritma dan struktur data. Pendekatan ini bekerja dengan cara membuat keputusan terbaik pada setiap langkah berdasarkan kondisi saat itu, dengan harapan bahwa keputusan tersebut akan menghasilkan solusi optimal secara keseluruhan.

Pada dasarnya, algoritma greedy membangun solusi secara bertahap, dimulai dari kondisi awal, lalu secara berurutan memilih elemen yang memberikan keuntungan terbesar atau hasil paling optimal. Proses ini terus dilakukan hingga seluruh elemen telah diproses atau solusi yang diinginkan tercapai.

Yang membedakan algoritma greedy dari pendekatan lain adalah sifatnya yang tidak meninjau kembali keputusan yang telah dibuat. Sekali sebuah pilihan diambil, maka dianggap final dan tidak akan diubah di langkah berikutnya. Karena itu, algoritma greedy hanya dapat digunakan pada jenis masalah tertentu yang memiliki dua karakteristik penting.

Karakteristik pertama adalah greedy choice property, yaitu masalah memungkinkan solusi optimal dibangun dari serangkaian pilihan lokal terbaik. Karakteristik kedua adalah optimal substructure, yaitu solusi optimal dari suatu masalah dapat dibentuk dari solusi optimal sub masalahnya.

Beberapa contoh penerapan algoritma greedy antara lain adalah pada masalah penukaran uang dengan pecahan tertentu (coin change problem), penjadwalan aktivitas (activity selection), serta pencarian minimum spanning tree seperti pada algoritma Kruskal dan Prim.

Meskipun algoritma greedy tergolong sederhana dan efisien, pendekatan ini tidak selalu menghasilkan solusi optimal untuk semua jenis masalah. Oleh karena itu, penting untuk terlebih dahulu menganalisis apakah karakteristik suatu masalah sesuai dengan prinsip greedy sebelum menggunakan pendekatan ini.

2.2 Cara Kerja Program

Program bot pada permainan Diamonds beroperasi melalui serangkaian komunikasi dengan sistem backend menggunakan protokol HTTP. Setiap tindakan yang dilakukan bot, mulai dari proses registrasi hingga pengambilan keputusan selama permainan, dilakukan dengan mengirim permintaan (request) ke berbagai endpoint API yang telah disediakan. Alur kerja program ini dijelaskan sebagai berikut:

1. Verifikasi Keberadaan Bot

   Langkah awal yang dilakukan program adalah memverifikasi apakah bot telah terdaftar dalam sistem. Hal ini dilakukan dengan mengirim permintaan POST ke endpoint /api/bots/recover yang menyertakan alamat email serta kata sandi bot. Apabila bot telah terdaftar, sistem akan merespons dengan kode status 200 dan mengirimkan ID bot. Sebaliknya, apabila data tidak ditemukan, server akan mengembalikan kode status 404.

2. Proses Registrasi Bot

   Jika bot belum terdaftar, maka proses registrasi akan dilakukan secara otomatis. Program mengirimkan POST request ke endpoint /api/bots dengan menyertakan informasi yang diperlukan, seperti nama bot, email, kata sandi, dan nama tim. Apabila pendaftaran berhasil, server akan memberikan respons berupa ID unik yang digunakan untuk identifikasi bot selama permainan.

3. Bergabung ke Dalam Papan Permainan (Board)

   Setelah ID bot diperoleh, langkah selanjutnya adalah memasukkan bot ke dalam papan permainan yang tersedia. Proses ini dilakukan dengan mengirimkan POST request ke /api/bots/{id}/join, yang di dalamnya mencantumkan ID papan permainan (board ID) yang dituju. Jika berhasil, sistem akan merespons dengan detail kondisi papan permainan yang akan digunakan oleh bot.

4. Pelaksanaan Aksi Bot Selama Permainan

   Program bot secara berkala melakukan perhitungan untuk menentukan arah gerak yang optimal berdasarkan kondisi papan permainan saat itu. Setiap keputusan gerakan dikirim ke backend melalui endpoint /api/bots/{id}/move dengan menyertakan nilai parameter arah, seperti “NORTH”, “SOUTH”, “EAST”, atau “WEST”. Respons dari server akan berisi pembaruan informasi kondisi papan setelah pergerakan tersebut dilakukan. Proses ini berulang selama permainan masih berlangsung.

5. Pembaruan Visualisasi Permainan di Antarmuka Pengguna

   Untuk memastikan visualisasi permainan selalu menampilkan kondisi terkini, sistem frontend secara berkala mengirimkan GET request ke endpoint /api/boards/{id} guna memperoleh data terbaru mengenai keadaan papan permainan. Dengan demikian, informasi yang ditampilkan kepada pemain selalu terjaga keakuratannya secara waktu nyata (real-time).

Seluruh rangkaian proses ini akan terus dijalankan secara otomatis oleh program hingga permainan selesai atau waktu permainan habis. Ketika waktu berakhir, sistem akan menghentikan bot dari permainan secara otomatis.

2.3 Cara Implementasi Program

Program ini dirancang untuk menerapkan kecerdasan buatan (Artificial Intelligence) melalui pendekatan algoritma greedy guna mengendalikan perilaku bot dalam permainan Diamonds. Mekanisme pengambilan keputusan dilakukan secara dinamis di setiap giliran permainan, di mana bot secara langsung memilih tindakan yang memberikan keuntungan terbesar saat itu, tanpa melakukan analisis jangka panjang terhadap konsekuensi dari tindakan tersebut.

Implementasi logika bot dikembangkan dalam sebuah kelas bernama StimananaLogic yang merupakan subclass dari BaseLogic. Metode inti dari program ini adalah next_move(), yang dipanggil secara otomatis oleh engine permainan pada setiap giliran untuk menentukan langkah bot selanjutnya.

Dalam proses pengambilan keputusan, prioritas aksi ditentukan berdasarkan kondisi permainan. Misalnya, jika bot sedang membawa tiga atau lebih diamond dan terdapat musuh di sekitar, maka keputusan yang diambil adalah kembali ke markas untuk mengamankan diamond. Namun, apabila jumlah diamond yang dibawa masih di bawah tiga, maka bot cenderung memilih untuk menekan tombol merah (red button), sesuai strategi yang ditentukan.

Untuk mendukung logika utama, digunakan beberapa fungsi tambahan seperti get_closest_diamond untuk mencari diamond terdekat, should_press_red_button untuk memutuskan penggunaan red button, serta distance_with_teleporter yang menghitung jarak dengan mempertimbangkan keberadaan teleporter. Seluruh fungsi dan logika program dikemas dalam satu berkas Python bernama stimanana.py, yang dijalankan melalui perintah python main.py sesuai dengan arsitektur framework yang digunakan.

2.4 Menjalankan Bot Program

Untuk menjalankan bot dalam permainan Diamonds, dilakukan beberapa tahap pengaturan sebagai berikut:

1. Penempatan Berkas Script Bot

   Langkah awal adalah menempatkan file Python berisi logika bot, yang dinamai stimana.py, ke dalam direktori game/logic pada struktur proyek permainan. File ini menjadi pusat pengendali perilaku bot dalam permainan, seperti navigasi, pengumpulan diamond, serta pengambilan keputusan strategis.
2. Pengecekan Dependensi dan Modul

   Bot memerlukan sejumlah modul agar dapat berfungsi dengan baik. Modul-modul seperti GameObject, Board, dan Position, serta fungsi tambahan seperti get_direction, harus terimpor dengan benar agar bot mampu mengenali kondisi permainan dan menentukan arah pergerakan secara tepat.
3. Integrasi Bot dengan Sistem Permainan

   Setelah file dan dependensi siap, bot harus diregistrasikan ke dalam sistem permainan agar bisa ikut serta dalam sesi permainan. Proses ini melibatkan penambahan bot ke

daftar peserta melalui konfigurasi khusus yang tersedia pada sistem atau framework game.

4. Eksekusi Permainan

   Untuk memulai permainan, digunakan command line atau terminal. Perintah yang dijalankan umumnya berupa python main.py atau disesuaikan dengan sistem yang dipakai. Setelah dijalankan, sistem akan otomatis mengeksekusi fungsi next_move() setiap giliran untuk menentukan pergerakan bot.

5. Observasi Jalannya Permainan

   Selama permainan berlangsung, bot akan bergerak mengikuti strategi yang telah ditentukan. Misalnya, bot akan memprioritaskan diamond bernilai tinggi yang lokasinya dapat dijangkau dengan efisien, menghindari lawan di jalur berisiko, memanfaatkan teleporter, dan kembali ke base saat diperlukan.

6. Tahap Pengujian dan Evaluasi Strategi

   Setelah bot dijalankan, dilakukan beberapa simulasi untuk melihat apakah perilakunya sudah sesuai dengan strategi greedy yang dirancang. Evaluasi dilakukan berdasarkan performa bot, lalu diperbaiki jika ada keputusan yang belum optimal atau menimbulkan risiko tinggi terhadap kehilangan diamond.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses Mapping

Dalam permainan Diamonds, terdapat berbagai elemen penting yang perlu dikenali oleh bot, seperti pemain (bot), markas (base), batu permata (diamond), serta objek spesial seperti teleporter dan red button. Bot yang dikembangkan akan membaca kondisi permainan secara real-time setiap giliran. Informasi ini kemudian digunakan untuk menentukan keputusan seperti apakah harus mencari diamond, kembali ke markas, atau memanfaatkan teleporter. Bot hanya akan mencari diamond jika inventaris belum penuh dan waktu masih cukup, dan akan memilih jalur tercepat, baik secara langsung maupun melalui teleporter.

Komponen strategi greedy yang diterapkan mencakup:

1. Kandidat Solusi – seluruh kemungkinan langkah yang dapat diambil bot pada setiap giliran.
2. Solusi Sementara – kumpulan tindakan dari himpunan kandidat yang menghasilkan poin terbanyak.
3. Fungsi Evaluasi Solusi – untuk menentukan apakah inventaris sudah penuh.
4. Fungsi Seleksi – memilih diamond berdasarkan density (rasio nilai terhadap jarak).
5. Fungsi Kelayakan – mengevaluasi apakah diamond dapat diamankan berdasarkan jarak, waktu tempuh, dan kecepatan bot musuh.
6. Fungsi Tujuan (Objektif) – memaksimalkan poin total yang diperoleh bot selama permainan.

3.2 Eksplorasi Strategi Alternatif Berbasis Greedy

Dalam tahapan eksplorasi strategi, dikaji beberapa pendekatan greedy yang berfokus pada efisiensi dalam pengambilan diamond serta pengembalian ke base. Pendekatan greedy yang diuji tidak hanya mempertimbangkan *jarak langsung* ke diamond, namun juga memasukkan

kondisi-kondisi dinamis seperti penggunaan teleporter, kapasitas inventory, dan situasi ketika bot mengalami stuck.

Berikut beberapa strategi yang dieksplorasi dan diuji melalui implementasi:

- Greedy by Nearest Diamond
  Strategi ini merupakan bentuk greedy konvensional di mana bot selalu bergerak menuju diamond terdekat berdasarkan jarak Manhattan. Ini menjadi dasar logika dalam mencari *target utama*.
- Greedy by Return to Base
  Jika inventory bot sudah penuh (misalnya memuat 5 diamond), maka bot langsung memprioritaskan kembali ke base untuk mengamankan poin, alih-alih mencari diamond lagi.
- Greedy by Escape (Saat Stuck)
  Jika bot terdeteksi tidak berpindah posisi selama beberapa langkah (lebih dari 2 langkah tetap di posisi yang sama), maka diasumsikan bot sedang mengalami stuck (misal: dihadang tembok atau musuh). Dalam kondisi ini, bot diarahkan kembali ke base sebagai langkah penyelamatan.
- Greedy by Teleport Optimization
  Saat ada dua teleporter di peta, bot akan mempertimbangkan apakah mengambil rute melalui teleporter akan menghasilkan jarak efektif yang lebih pendek menuju target diamond. Jika ya, maka bot akan bergerak menuju teleporter terlebih dahulu.
- Greedy by Base Proximity
  Saat bot sedang membawa diamond namun jarak ke base lebih dekat dibanding jarak ke diamond terdekat, maka bot akan diarahkan kembali ke base, meskipun inventory belum penuh. Ini bertujuan untuk mengurangi risiko kehilangan poin.

3.3 Implementasi Strategi Greedy pada Bot

Strategi greedy diterapkan dalam kelas MyBot, yang diturunkan dari BaseLogic. Bot ini mengmbil keputusan berdasarkan kondisi terbaik di setiap langkah, tanpa mempertimbangkan hasil jangka panjang secara keseluruhan.a

Logika utama strategi ini meliputi:

1. Mencari Diamond Terdekat
   Bot selalu mengincar diamond dengan jarak terpendek berdasarkan jarak Manhattan dari posisinya saat ini.

2. Kembali ke Base Saat Inventory Penuh
   Ketika jumlah diamond mencapai batas maksimum (5), bot langsung diarahkan kembali ke base untuk menyetorkan hasil.

3. Prioritas Kembali ke Base
   Jika bot membawa diamond dan jarak ke base lebih dekat dibandingkan diamond terdekat, maka bot lebih dulu kembali ke base.

4. Penggunaan Teleporter
   Jika tersedia dua teleporter, bot menghitung apakah penggunaan teleporter lebih menguntungkan dalam mencapai diamond, dan akan memanfaatkannya bila iya.

5. Deteksi dan Penanganan Stuck
   Bot mendeteksi kondisi "stuck" jika posisinya tidak berubah selama tiga langkah, dan akan kembali ke base untuk mereset pergerakan.

6. Pemilihan Arah Gerak
   Gerakan diprioritaskan berdasarkan sumbu dengan jarak terbesar ke target, untuk memastikan langkah yang efisien.

Dengan menggabungkan strategi greedy dan logika adaptif terhadap kondisi permainan (seperti teleporter dan stuck), bot ini dapat bergerak secara efektif dan responsif terhadap perubahan situasi.

3.4 Strategi Greedy yang Dipilih

Dalam pengembangan bot *StimananaLogic*, strategi greedy dipilih karena kemampuannya menghasilkan keputusan cepat dan efisien dalam permainan real-time. Strategi utama yang digunakan adalah Greedy by Density, yaitu memilih diamond berdasarkan rasio nilai poin terhadap jarak (density). Semakin tinggi density-nya, semakin tinggi prioritasnya untuk diambil.

Untuk menyesuaikan dengan dinamika permainan, strategi utama ini dilengkapi dengan beberapa sub-strategi adaptif:

a. Greedy by Escape
   Bot segera kembali ke base saat membawa ≥3 diamond dan musuh berada dalam jarak dekat.
b. Greedy by Return (Waktu)
   Bot pulang jika waktu tersisa hanya cukup untuk mencapai base.
c. Greedy by Tackle
   Bot menyerang musuh terdekat jika membawa ≥2 diamond, untuk mencuri diamond lawan.
d. Greedy by Red Button
   Bot menuju red button jika jumlah diamond tersisa di peta sedikit, untuk me-reset dan menambah diamond.
e. Greedy by Red Diamond
   Diamond merah (2 poin) diutamakan meski sedikit lebih jauh, selama inventory belum penuh.
f. Greedy by Distance & Mampir ke Base
   Bot pulang saat inventory penuh atau saat perjalanan melewati base, untuk efisiensi langkah dan penyimpanan.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

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

class MyBot(BaseLogic):
   def __init__(self):
       super().__init__()
       self.last_position = None
       self.back_to_base = False
       self.target_diamond = None
       self.stuck_count = 0
       self.base_position = None
       self.teleporters = []
       self.visited_positions = set()

   def next_move(self, board_bot: GameObject, board: Board) -> tuple[int, int]:
       # Helper function: Manhattan distance
       def manhattan_distance(pos1: Position, pos2: Position) -> int:
           return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

       # Helper function: Move to target
       def move_to_target(current: Position, target: Position) -> tuple[int,
int]:
           if current.x == target.x and current.y == target.y:
               return (0, 0)

           # Handle teleporter
           if len(teleporters) == 2:
               tele1, tele2 = teleporters
               if current.x == tele1.x and current.y == tele1.y:
                   target = tele2
               elif current.x == tele2.x and current.y == tele2.y:
                   target = tele1

           dx = 0
           dy = 0

           if current.x < target.x:
```

```
                dx = 1
            elif current.x > target.x:
                dx = -1

            if current.y < target.y:
                dy = 1
            elif current.y > target.y:
                dy = -1

            x_dist = abs(current.x - target.x)
            y_dist = abs(current.y - target.y)

            if x_dist > y_dist and dx != 0:
                return (dx, 0)
            elif dy != 0:
                return (0, dy)
            elif dx != 0:
                return (dx, 0)

            return (0, 0)

        # Helper function: Calculate effective distance
        def calculate_effective_distance(start: Position, end: Position) -> int:
            if len(teleporters) != 2:
                return manhattan_distance(start, end)

            tele1, tele2 = teleporters
            options = [
                manhattan_distance(start, end),
                manhattan_distance(start, tele1) + manhattan_distance(tele2,
end),
                manhattan_distance(start, tele2) + manhattan_distance(tele1,
end)
            ]
            return min(options)

        # Helper function: Find closest diamond
        def find_closest_diamond(from_pos: Position) -> tuple[Position, float]:
            closest = None
            min_distance = float('inf')

            for obj in board.game_objects:
                if obj.type == "DiamondGameObject":
                    distance = manhattan_distance(from_pos, obj.position)
                    if distance < min_distance:
                        min_distance = distance
                        closest = obj.position
```

```
            return closest, min_distance

        # Helper function: Check if stuck
        def check_if_stuck(current_pos: Position) -> bool:
            nonlocal last_position, stuck_count
            if last_position and last_position.x == current_pos.x and
last_position.y == current_pos.y:
                stuck_count += 1
            else:
                stuck_count = 0
            last_position = current_pos
            return stuck_count > 2

        # Helper function: Handle stuck situation
        def handle_stuck_situation(current_pos: Position) -> tuple[int, int]:
            nonlocal back_to_base, stuck_count
            back_to_base = True
            stuck_count = 0
            return move_to_target(current_pos, base_position)

        # Main logic
        current_pos = board_bot.position
        props = board_bot.properties
        base_position = props.base
        last_position = self.last_position
        stuck_count = self.stuck_count
        back_to_base = self.back_to_base
        teleporters = [obj.position for obj in board.game_objects if obj.type ==
"TeleportGameObject"]
        self.visited_positions.add((current_pos.x, current_pos.y))

        # Check if stuck
        if check_if_stuck(current_pos):
            return handle_stuck_situation(current_pos)

        # Jika inventory penuh, langsung kembali ke base
        if props.diamonds == 5:
            return move_to_target(current_pos, base_position)

        # Hitung jarak ke base
        base_distance = calculate_effective_distance(current_pos, base_position)

        # Cari diamond terdekat
        closest_diamond, closest_diamond_distance =
find_closest_diamond(current_pos)
```

```
        # Jika sedang membawa diamond dan base lebih dekat, utamakan kembali ke
base
        if props.diamonds > 0 and base_distance < closest_diamond_distance:
            return move_to_target(current_pos, base_position)

        # Jika ada diamond yang bisa diambil
        if closest_diamond:
            # Cek apakah melalui teleporter lebih efisien
            if len(teleporters) == 2:
                tele1, tele2 = teleporters

                via_tele1 = (manhattan_distance(current_pos, tele1) +
                             manhattan_distance(tele2, closest_diamond))

                via_tele2 = (manhattan_distance(current_pos, tele2) +
                             manhattan_distance(tele1, closest_diamond))

                if via_tele1 < closest_diamond_distance and via_tele1 <
via_tele2:
                    return move_to_target(current_pos, tele1)
                elif via_tele2 < closest_diamond_distance:
                    return move_to_target(current_pos, tele2)

            return move_to_target(current_pos, closest_diamond)

        # Default: kembali ke base
        return move_to_target(current_pos, base_position)
```

2. Penjelasan Alur Program

Program di atas merupakan implementasi logika dari sebuah bot bernama MyBot yang diturunkan dari kelas BaseLogic. Bot ini dirancang untuk memainkan game berbasis grid, di mana tujuan utamanya adalah mengumpulkan diamond sebanyak mungkin dan membawanya kembali ke markas (base). Saat bot diinisialisasi melalui konstruktor __init__, sejumlah atribut disiapkan, seperti posisi terakhir (last_position), status apakah sedang kembali ke base (back_to_base), target diamond yang sedang dikejar (target_diamond), jumlah langkah diam (stuck count), posisi base, daftar teleporter, dan kumpulan posisi yang telah dikunjungi

(visited_positions). Atribut-atribut ini berguna untuk membantu pengambilan keputusan saat game berlangsung.

Logika utama dijalankan dalam metode next_move, yang akan menentukan arah gerakan bot pada setiap giliran berdasarkan kondisi papan permainan (board) dan informasi bot (board_bot). Di dalam metode ini terdapat beberapa fungsi pembantu. Fungsi manhattan_distance menghitung jarak antara dua titik secara horizontal dan vertikal (tanpa diagonal), sedangkan move_to_target mengarahkan bot menuju suatu posisi target dengan mempertimbangkan penggunaan teleporter jika tersedia dan lebih efisien. Jika bot sedang berada di salah satu dari dua teleporter, maka target akan otomatis diarahkan ke pasangan teleporternya. Fungsi calculate_effective_distance digunakan untuk menghitung jarak tercepat dari satu titik ke titik lain, baik langsung maupun melalui teleportasi.

Selanjutnya, fungsi find_closest_diamond mencari diamond terdekat dari posisi saat ini berdasarkan jarak Manhattan, dan check_if_stuck mendeteksi apakah bot terjebak di tempat yang sama selama beberapa giliran. Jika iya, maka fungsi handle_stuck_situation akan memaksa bot untuk kembali ke base sebagai bentuk "reset".

Dalam logika utama, bot pertama-tama menyimpan data posisi saat ini dan memperbarui daftar teleporter. Jika bot terdeteksi sedang stuck (diam di tempat lebih dari dua giliran), maka bot langsung bergerak menuju base untuk menghindari kebuntuan. Apabila inventory bot sudah penuh (memiliki 5 diamond), maka bot juga langsung kembali ke base tanpa mencari diamond lagi. Jika inventory belum penuh, maka bot akan membandingkan jarak ke base dan ke diamond terdekat. Bila bot sedang membawa diamond dan base lebih dekat dibanding diamond berikutnya, maka bot diprioritaskan untuk kembali ke base terlebih dahulu. Namun, jika belum membawa diamond, bot akan menuju diamond terdekat. Dalam hal ini, jika perjalanan melalui teleport lebih efisien dibanding langsung, maka bot akan diarahkan menuju teleporter terlebih dahulu.

Jika tidak ada diamond tersisa, maka bot secara default akan kembali ke base. Dengan pendekatan ini, MyBot menerapkan strategi greedy yang selalu mencari jalan tercepat ke diamond atau base, sambil menangani kondisi seperti macet dan teleportasi untuk efisiensi perjalanan.

4.2 Struktur Data yang Digunakan

Dalam program MyBot, berbagai struktur data digunakan untuk mendukung pengambilan keputusan bot selama permainan. Struktur data utama yang digunakan adalah objek-objek yang berasal dari kelas seperti Position, GameObject, dan Board. Objek Position merepresentasikan titik koordinat dalam papan permainan dan digunakan untuk menunjuk lokasi bot, diamond, base, maupun teleporter. Objek GameObject digunakan untuk merepresentasikan semua entitas di papan, termasuk diamond dan teleport, lengkap dengan informasi jenis objek dan posisinya. Sedangkan Board menyimpan seluruh informasi kondisi permainan saat ini, termasuk daftar objek yang ada di papan.

Selain objek, program juga menggunakan beberapa struktur data bawaan Python. Misalnya, list digunakan untuk menyimpan daftar teleporters yang ditemukan di papan. Kemudian, set digunakan pada atribut visited_positions untuk menyimpan kumpulan koordinat yang telah dikunjungi bot agar tidak terjadi pengulangan gerakan. Tipe data tuple juga banyak digunakan untuk mengembalikan arah gerak bot berupa pasangan (dx, dy) yang menunjukkan perpindahan arah pada sumbu x dan y. Selain itu, nilai-nilai seperti posisi terakhir (last_position), target diamond (target_diamond), dan base position disimpan dalam variabel bertipe Position, sementara variabel seperti stuck_count, back_to_base, dan diamonds menggunakan tipe data primitif seperti int dan bool. Penggunaan struktur data ini memungkinkan bot untuk menyimpan konteks permainan, menavigasi papan dengan efisien, dan merespons situasi seperti teleportasi dan kemacetan (stuck) dengan tepat.

4.3 Pengujian Program

4.3.1 Skenario Pengujian

Pengujian dilakukan untuk memastikan bahwa bot MyBot dapat menjalankan fungsinya sesuai dengan logika yang telah dirancang, yaitu mengumpulkan diamond secara efisien dan kembali ke base dalam kondisi optimal. Beberapa skenario pengujian dirancang untuk menguji perilaku bot dalam berbagai kondisi, antara lain:

- **Bot mengambil diamond dan kembali ke base saat inventory penuh**

Papan permainan berisi beberapa diamond dan base berada dalam jarak sedang. Bot diuji apakah ia kembali ke base setelah mengumpulkan lima diamond.

- **Bot memilih diamond terdekat**

Terdapat beberapa diamond dengan jarak yang berbeda. Bot diuji apakah ia selalu menuju diamond dengan jarak terpendek.

- **Bot memanfaatkan teleport untuk rute lebih efisien**

Diletakkan dua buah teleport dengan posisi yang memungkinkan rute menuju diamond menjadi lebih pendek. Bot diuji apakah ia melewati teleport jika itu lebih menguntungkan.

- **Bot mendeteksi kondisi stuck (diam di posisi yang sama lebih dari 2 kali)**

Diuji dengan kondisi bot sengaja terjebak atau diarahkan ke posisi yang membuatnya tidak bisa bergerak. Bot diuji apakah ia mendeteksi kebuntuan dan mengambil tindakan untuk kembali ke base.

- **Bot kembali ke base lebih dulu jika base lebih dekat dibandingkan diamond berikutnya saat sudah membawa diamond**

Dalam kondisi bot membawa 1–4 diamond dan ada diamond lain yang lebih jauh daripada base, bot diuji apakah ia memilih kembali ke base.

4.3.2 Hasill Pengujian dan Analisis

Dari hasil pengujian berdasarkan skenario yang dirancang, diperoleh data sebagai berikut:

1. **Bot kembali ke base saat inventory penuh**
    - Hasil: Sesuai. Bot langsung menuju base setelah membawa 5 diamond.
    - Analisis: Logika pengecekan if props.diamonds == 5: bekerja dengan baik.
2. **Bot memilih diamond terdekat**
    - Hasil: Sesuai. Bot selalu menuju diamond dengan jarak Manhattan terkecil.
    - Analisis: Fungsi find_closest_diamond berhasil mengidentifikasi dan mengutamakan diamond terdekat.
3. **Bot memanfaatkan teleportasi**

- Hasil: Sesuai. Ketika rute melalui teleport lebih pendek, bot memilih mendekati teleporter.
- Analisis: Perhitungan via_tele1 dan via_tele2 bekerja dengan baik untuk mengevaluasi efisiensi rute.

4. **Bot mendeteksi stuck dan keluar dari kondisi tersebut**
    - Hasil: Sesuai. Setelah terdeteksi diam di tempat selama 3 giliran, bot mengubah arah menuju base.
    - Analisis: Fungsi check_if_stuck dan handle_stuck_situation berhasil dijalankan sesuai kondisi.
5. **Bot memilih kembali ke base jika base lebih dekat daripada diamond**
    - Hasil: Sesuai. Dalam beberapa percobaan, bot lebih memilih kembali ke base daripada mengejar diamond yang lebih jauh.
    - Analisis: Logika perbandingan if props.diamonds > 0 and base_distance < closest_diamond_distance: sudah tepat.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Strategi greedy terbukti efektif dalam pengambilan keputusan cepat pada bot permainan Diamonds. Bot dapat memilih langkah berdasarkan kondisi seperti jarak ke diamond, kapasitas inventory, dan posisi base, serta memanfaatkan teleporter untuk efisiensi. Strategi utama berbasis density diamond terhadap jarak memungkinkan bot memilih target yang menguntungkan secara efisien.

Sub-strategi seperti escape, return, dan penggunaan teleporter sudah berjalan dengan baik, meskipun beberapa strategi seperti tackle dan red button belum sepenuhnya diimplementasikan. Struktur data dan fungsi yang dipakai mendukung proses pengambilan keputusan secara efisien dan fleksibel, sehingga memudahkan pengembangan lebih lanjut.

5.2 Saran

1. Disarankan untuk menambahkan mekanisme deteksi serta strategi penanganan terhadap keberadaan musuh, agar bot dapat melakukan serangan atau penghindaran secara lebih efektif.
2. Perlu diimplementasikan strategi penggunaan tombol merah (red button) secara dinamis agar bot dapat memaksimalkan pengumpulan diamond yang tersedia.
3. Penghitungan nilai density sebaiknya diperbaiki dengan memasukkan faktor nilai poin dari diamond, sehingga pengambilan keputusan menjadi lebih optimal.
4. Disarankan untuk mengembangkan bot dengan metode pembelajaran mesin guna meningkatkan kemampuan adaptasi terhadap berbagai situasi dan lawan.
5. Perlu dilakukan pengujian yang lebih komprehensif untuk mengevaluasi kinerja bot dan melakukan penyempurnaan strategi secara menyeluruh.

LAMPIRAN

A. **Repository Github (https://github.com/RSANDI-p/TUBES1_Algoventure.git)**

B. **Video Penjelasan (https://youtu.be/6cZFpEHN1t4)**