# IMPLEMENTASI ALGORITMA GREEDY RATIO DAN BFS DALAM PEMECAHAN BOT PERMAINAN DIAMOND

## Tugas Besar

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


**Oleh: Kelompok MpokZen**

| | |
|---|---|
| Romualdus Hary Prabowo | 123140083 |
| Juliani Leony Putri Melati Manalu | 123140029 |
| Muhammad Fadhel | 123140106 |

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

Tugas besar strategi algoritma mengambil sebuah studi kasus permainan Diamonds, sebuah programming challenge yang mempertandingkan kecerdasan buatan (bot) buatan setiap kelompok. Setiap kelompok diminta untuk membuat sebuah bot yang mampu bergerak secara otomatis dan strategis dalam lingkup permainan untuk mengumpulkan diamond sebanyak - banyaknya dengan waktu yang tersedia.

Permainan Diamonds mengandung elemen yang kompleks seperti peta acak, regenerasi diamond, teleportasi, dan interaksi antar bot (tackle), sehingga setiap kelompok diminta untuk dapat menyusun sebuah strategi algoritma terbaik dari algoritma greedy yang sudah dipelajari pada perkuliahan Strategi Algoritma. Tujuan utama dari pengimplementasian algoritma greedy pada bot adalah untuk memperoleh skor sebanyak - banyaknya dengan mengumpulkan sebuah diamond yang tersedia secara acak. Diamond biru bernilai satu, dan diamond merah bernilai dua Bot memiliki inventory untuk penyimpanan diamond sementara, yang terbatas maksimal untuk lima diamond. Jika inventory penuh, maka bot harus kembali ke base untuk meletakan diamond - diamond yang sudah diambil.

Setiap kelompok harus mampu membuat sebuah bot yang dapat:

1. Navigasi peta
2. Mengambil keputusan cepat berdasarkan kondisi yang ada
3. Kembali ke base untuk menyimpan diamond ketika inventory sudah penuh
4. Menghindari interaksi negatif seperti tackle dari bot lawan
5. Memanfaatkan fitur permainan seperti teleporter untuk berpindah posisi, dan red button untuk keuntungan strategis (Diamond dibuat ulang secara acak).

Permainan Diamonds dapat dijalankan dengan menggunakan game engine dan bot starter pack yang telah disediakan. Game engine membuat sebuah logika backend dan visualisasi permainan. Starter pack berisi sebuah kode awal untuk melakukan akses API dan struktur dasar bot. Strategi utama yang digunakan dalam permainan ini adalah strategi greedy, yang harus diimplementasikan oleh setiap kelompok dengan kebebasan dalam perencanaan strategi yang ini digunakan.

# BAB II
# LANDASAN TEORI

## 2.1 Dasar Teori

Algoritma greedy adalah suatu metode penyelesaian masalah yang mengambil keputusan terbaik pada setiap langkah lokal dengan harapan solusi tersebut akan menghasilkan solusi global yang optimal [1]. Pendekatan ini tidak meninjau semua kemungkinan solusi secara menyeluruh, melainkan memilih solusi parsial yang tampak paling menguntungkan saat itu juga, kemudian melanjutkan proses hingga mencapai akhir masalah. Meskipun tidak selalu memberikan solusi optimal secara global, algoritma greedy sering kali cukup efisien dan mudah diimplementasikan, serta cocok untuk permasalahan dengan sifat *greedy-choice property* dan *optimal substructure* [2].

Dalam implementasinya, algoritma greedy memiliki berbagai bentuk dan variasi strategi, termasuk Greedy Ratio dan Breadth-First Search (BFS).

1. Breadth-First Search (BFS) adalah algoritma penelusuran graf yang menelusuri simpul-simpul berdasarkan level kedalaman, dimulai dari node akar dan mengeksplorasi semua tetangga pada level tersebut sebelum berpindah ke level berikutnya. BFS sangat berguna untuk mencari jalur terpendek dalam graf tak berbobot [3]. Dalam permainan *Diamonds*, algoritma BFS digunakan oleh bot untuk mencari jalur terdekat dari posisi saat ini menuju tujuan seperti diamond atau base.
2. Strategi Greedy Ratio adalah bentuk penerapan greedy yang mempertimbangkan perbandingan antara nilai yang didapat dengan biaya yang dikeluarkan, sehingga keputusan diambil berdasarkan rasio efisiensi tersebut. Misalnya, dalam permainan *Diamonds*, jika terdapat dua diamond yaitu:
    - Diamond merah: bernilai 2 poin, jarak 5 langkah → rasio = 2/5 = 0.4
    - Diamond biru: bernilai 1 poin, jarak 1 langkah → rasio = 1/1 = 1.0

Dengan strategi greedy ratio, bot akan memilih diamond biru karena memberikan nilai lebih tinggi per langkah yang ditempuh. Strategi ini berguna untuk memaksimalkan skor dalam waktu dan ruang gerak yang terbatas. Pendekatan greedy ini diimplementasikan sebagai bagian dari strategi bot dalam permainan *Diamonds*, di mana bot dirancang untuk mengumpulkan diamond sebanyak-banyaknya sambil menghindari risiko seperti inventory penuh atau ditackle oleh bot lawan. Oleh karena itu, penggunaan algoritma greedy yang tepat sangat penting dalam menentukan efisiensi dan efektivitas strategi permainan.

## 2.2 Cara Kerja Program

Bot dirancang untuk dapat bergerak secara otomatis pada sebuah peta permainan yang tersedia, dan mengumpulkan diamond yang tersebar secara acak dengan cara bergerak menuju diamond - diamond tersebut sesuai dengan algoritma yang telah diterapkan pada bot. Bot akan terus bergerak mengumpulkan diamond yang tersebar dan kembali pada base jika inventory yang dimiliki telah mencapai kapasitas maksimalnya hingga waktu permainan berakhir. Bot yang berhasil mengumpulkan diamond sebanyaknya pada base hingga waktu permainan berakhir, maka bot tersebut yang menjadi pemenang.

1. **Cara Implementasi Program**

   Implementasi program dapat dilakukan dengan menentukan strategi greedy yang hendak diterapkan pada bot yang dimiliki, untuk nantikan dilakukan pengkodingan dengan bahasa python terhadap strategi tersebut. Pada kasus ini, strategi yang kami gunakan diantaranya yaitu:

   - BFS. Strategi BFS kami gunakan untuk mengeksplorasi jalur terdekat ke diamond yang ada [2], [4].
   - Ratio. Strategi ratio kami gunakan untuk menentukan rasio nilai diamond terhadap jarak perjalanan menuju diamond tersebut [5], [6].

   Kedua strategi tersebut kami implementasikan dengan melakukan pengkodingan python untuk diimplementasikan pada bot kami. Dengan pengkodingan berdasarkan strategi yang kami pilih, bot yang kami miliki akan mampu melihat, dan memperhitungkan diamond mana yang hendak diambil berdasarkan nilai dan biaya perjalanan yang dibutuhkan untuk sampai pada tujuan diamond.Jika kapasitas inventory yang ada pada bot telah penuh, bot akan kembali pada base sebelum nantinya akan kembali berjalan mencari diamond - diamond yang ada sesuai logika strategi yang telah diterapkan.

2. **Menjalankan Bot Program**

   Program bot dapat dijalankan dengan mengeksekusi skrip python utama yang telah dibuat sebelumnya berdasarkan strategi - strategi yang telah dipilih untuk diterapkan. Bot akan bergerak sesuai strategi, dan logika yang ada pada skrip utama permainan. Peta dan status permainan dapat diperoleh melalui rikues API yang tersedia pada tugas besar strategi algoritma berikut ini.

# BAB III
# APLIKASI STRATEGI *GREEDY*

## 3.1 Proses *Mapping*

Proses Mapping adalah langkah pertama yang dilakukan oleh bot agar dapat memahami keadaan di lingkungan permainan, termasuk lokasi diamond, lokasi base, tempat teleporter, serta keberadaan bot lawan. Pemetaan ini dilakukan secara rutin dengan memanfaatkan data yang diperoleh dari API permainan, yang memberikan informasi lengkap terkait peta permainan dalam bentuk matriks [7].

Setiap sel pada peta menyimpan informasi tertentu, seperti:

- Diamond biru (nilai 1)
- Diamond merah (nilai 2)
- Base (tempat penyimpanan diamond)
- Teleporter (untuk berpindah tempat dengan cepat)
- Tombol merah (untuk mengacak posisi diamond)
- Bot musuh (yang bisa melakukan tackle)
- Dinding atau penghalang

Bot kemudian menyimpan informasi ini dalam struktur data peta internal yang berbentuk grid dua dimensi. Grid ini diperbaharui secara berkala setiap kali data peta diperoleh dari API, sehingga bot selalu memiliki informasi terkini.

Langkah-langkah dalam proses pemetaan yang dilaksanakan oleh bot kami mencakup:

1. Pengambilan data peta dari API

Bot mengakses titik akhir API yang memberikan gambaran peta terbaru dalam format JSON atau struktur data serupa.

2. Penguraian data peta

Data yang didapat diproses untuk mengenali tipe objek di setiap koordinat peta, seperti posisi diamond, base, teleporter, dan lain-lain.

3. Penyimpanan informasi dalam struktur data internal

Informasi yang telah diolah lalu disimpan dalam bentuk array atau kamus yang memungkinkan bot mengakses informasi secara efisien untuk pengambilan keputusan.

4. Visualisasi

Dalam fase pengujian, tim juga membuat representasi visual sederhana dari peta untuk membantu dalam proses debugging dan pengembangan strategi navigasi.

Proses pemetaan ini merupakan dasar yang sangat penting untuk semua pengambilan keputusan selanjutnya. Bot harus dengan tepat mengetahui posisinya dan objek penting di peta agar dapat menerapkan strategi greedy secara efektif [6].

### 3.2 Eksplorasi Alternatif Solusi Greedy

Dalam pengembangan strategi bot untuk permainan *Diamonds*, kami mengeksplorasi beberapa alternatif solusi berbasis algoritma greedy. Meskipun strategi utama yang ditugaskan adalah pendekatan greedy, terdapat berbagai cara dalam mengimplementasikannya, tergantung pada cara bot memprioritaskan keputusan dan menilai efisiensi pengambilan diamond. Berikut beberapa alternatif solusi greedy yang kami pertimbangkan:

1. Greedy Ratio dengan BFS

Strategi ini menghitung rasio antara nilai diamond dengan jarak yang dibutuhkan untuk mencapainya. Rasio dihitung sebagai: Rasio = Nilai diamond / Jarak
Jarak dihitung menggunakan algoritma Breadth-First Search (BFS), yang memberikan estimasi jalur terpendek ke tujuan. Diamond dengan rasio tertinggi akan diprioritaskan oleh bot. Strategi ini cukup efisien dalam kondisi di mana waktu terbatas dan diamond tersebar secara acak.

2. Greedy Murni Berbasis Nilai Tertinggi

Pada pendekatan ini, bot selalu mengejar diamond dengan nilai tertinggi tanpa memperhitungkan jarak. Misalnya, bot akan selalu mendahulukan diamond merah (2 poin) dibandingkan diamond biru (1 poin), walaupun lokasinya jauh. Pendekatan ini sederhana, namun sering kali tidak efisien karena dapat menghabiskan waktu untuk mengejar target yang sulit dijangkau [5].

3. Greedy Ratio dengan Dijkstra

Alternatif lain yang sempat kami pertimbangkan adalah memanfaatkan algoritma Dijkstra untuk menghitung jarak terpendek ke diamond, menggantikan BFS. Dijkstra lebih fleksibel untuk graf berbobot, sehingga cocok jika permainan kelak menerapkan sistem hambatan atau bobot pada jalur. Namun, karena permainan ini menggunakan peta grid sederhana tanpa bobot, penggunaan Dijkstra menjadi kurang efisien dibanding BFS [6].

### 3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setelah mengeksplorasi beberapa alternatif solusi greedy, kami melakukan analisis terhadap efisiensi dan efektivitas dari masing-masing pendekatan dalam konteks permainan *Diamonds*. Berikut hasil analisis kami:

1. Greedy Ratio dengan BFS

   **Efisiensi**: Pendekatan ini cukup efisien karena algoritma BFS mampu menemukan jalur terpendek dalam waktu yang relatif cepat pada peta grid tak berbobot. Dengan menghitung rasio antara nilai dan jarak, bot dapat menentukan target diamond yang memberikan keuntungan paling tinggi per langkah yang diambil [4].

   **Efektivitas**: Strategi ini efektif dalam memaksimalkan skor, terutama dalam waktu permainan yang terbatas. Dengan prioritas pada rasio tertinggi, bot cenderung lebih sering mendapatkan skor daripada hanya mengejar diamond bernilai tinggi yang letaknya jauh.

2. Greedy Murni Berbasis Nilai Tertinggi

   **Efisiensi**: Sederhana dan cepat dalam proses pengambilan keputusan karena hanya membandingkan nilai diamond. Namun, dapat menjadi tidak efisien dalam praktik, karena bot mungkin berjalan jauh untuk diamond merah sambil melewati diamond biru yang lebih dekat [5].

   **Efektivitas**: Rendah. Walaupun bot mengejar diamond bernilai tinggi, waktu yang terbuang untuk mencapai lokasi jauh justru dapat mengurangi total skor yang diperoleh selama permainan.

3. Greedy Ratio dengan Dijkstra

   **Efisiensi**: Dijkstra memberikan akurasi tinggi dalam menghitung jarak minimum pada graf berbobot. Namun, dalam peta permainan *Diamonds* yang tidak memiliki bobot jalur, algoritma ini kurang efisien karena prosesnya lebih kompleks dan memakan waktu dibandingkan BFS [6].

   **Efektivitas**: Tidak jauh berbeda dengan Greedy Ratio + BFS dalam konteks permainan ini. Oleh karena itu, penggunaan Dijkstra dianggap overkill.

Kesimpulan Analisis: Berdasarkan hasil analisis, strategi Greedy Ratio dengan BFS memberikan keseimbangan terbaik antara efisiensi dan efektivitas. Strategi ini tidak hanya memperhitungkan nilai diamond, tetapi juga mempertimbangkan jarak secara optimal dengan algoritma yang cepat dan ringan dijalankan.

### 3.4 Strategi Greedy yang Dipilih

Setelah melakukan eksplorasi berbagai alternatif solusi greedy dan menganalisis efisiensi serta efektivitasnya, kami memutuskan untuk menggunakan strategi Greedy Ratio dengan Breadth-First Search (BFS) sebagai pendekatan utama dalam pengembangan bot permainan *Diamonds*.

Strategi ini dipilih karena mampu menggabungkan dua aspek penting dalam permainan, yaitu:

- Jalur tercepat menuju target (berkat implementasi BFS yang efisien dalam graf tak berbobot)
- Pemilihan target yang menguntungkan secara nilai (melalui perhitungan rasio antara nilai diamond dan jaraknya)

Bot akan mengevaluasi setiap diamond yang terdeteksi berdasarkan rasio nilai terhadap jarak. Diamond dengan nilai tertinggi per satuan jarak akan diprioritaskan untuk diambil terlebih dahulu. Misalnya, jika tersedia diamond merah (nilai = 2, jarak = 5) dan diamond biru (nilai = 1, jarak = 1), maka bot akan memilih diamond biru karena memiliki rasio 1.0 dibandingkan rasio 0.4 untuk diamond merah. Pendekatan ini sesuai dengan prinsip greedy-choice property, yaitu memilih solusi lokal terbaik dengan harapan hasil global yang optimal [1] [2].

Untuk penentuan jalur menuju target yang dipilih, algoritma BFS digunakan agar bot dapat menemukan jalur terpendek secara efisien. BFS dinilai paling cocok karena peta permainan bersifat tidak berbobot dan berubah-ubah secara dinamis [3].
Selain itu, kami juga menambahkan logika untuk:

- Kembali ke base saat inventory penuh (maksimum 5 diamond)
- Memanfaatkan fitur seperti teleporter dan red button untuk efisiensi gerakan dan regenerasi diamond
- Menghindari tackle oleh bot lawan dengan tidak memilih jalur yang berisiko tinggi berdasarkan posisi musuh

Dengan mengintegrasikan strategi greedy ratio dan algoritma BFS, bot yang kami kembangkan memiliki kemampuan untuk mengambil keputusan secara cepat, efisien, dan adaptif terhadap perubahan peta permainan secara real-time. Pendekatan ini memberikan keseimbangan antara kesederhanaan algoritma dan efektivitas perolehan skor, sehingga menjadi pilihan optimal untuk tugas besar Strategi Algoritma ini.

# 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
from collections import deque

# Membuat class bot dengan nama MpokZen
class MpokZenBot(BaseLogic):
    def __init__(self):
        self.diamonds = [] # Tempat menyimpan lokasi diamond yg ada
          self.obstacles = set() # Tempat menyimpan rintangan yg ada
pada peta
    # fungsi mendapatkan objek peta
    def get_objects(self, board: Board):
        self.diamonds = [] # Tempat menyimpan diamond yg diambil
        self.obstacles = set() # Set obstacles untuk ditandai

        # Perulangan melihat kondisi/objek pada peta
        for object in board.game_objects:
                            if  object.type  in  ["DiamondGameObject",
"DiamondButtonGameObject"]: #Jika objek adalah diamon
                        self.diamonds.append(object) # Menyimpan posisi
diamond
            elif object.type in ["BotGameObject", "WallGameObject"]:
# Jika objek adalah rintangan
                                    self.obstacles.add((object.position.x,
object.eposition.y)) # Menyimpan posisi rintangan

    # Algoritma BFS. Mencari jarak terbaik
     def bfs(self, start: Position, goal: Position, board: Board) ->
int:
        visited = set()
        queue = deque([(start.x, start.y, 0)])
        visited.add((start.x, start.y))

        # Antrian untuk menjelajahi peta
        while queue:
            x, y, dist = queue.popleft()
            if (x, y) == (goal.x, goal.y):
                return dist

            for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
                nx, ny = x + dx, y + dy
                if 0 <= nx < board.width and 0 <= ny < board.height:
                        if (nx, ny) not in visited and (nx, ny) not in
self.obstacles:
```

```
                        queue.append((nx, ny, dist + 1))
                        visited.add((nx, ny))
        return float('inf')

    # Fungsi untuk menentukan langkah bot
    def next_move(self, board_bot: GameObject, board: Board):
        # Mengambil seluruh informasi pada peta
        self.get_objects(board)

        # Data bot
        my_pos = board_bot.position # Posisi bot saat ini
         inventory = board_bot.properties.diamonds # Deklarasi untuk
menyimpan diamon yg sudah diambil
            capacity = board_bot.properties.inventory_size # Untuk
kapasitas tas
             space_left = capacity - inventory # Menghitung sisa
kapasitas pada tas


        best_target = None
        best_ratio = -1
        best_distance = float('inf')

        # Algoritma ratio
        for diamond in self.diamonds:
                    if diamond.properties.points is not None and
diamond.properties.points <= space_left:
                dist = self.bfs(my_pos, diamond.position, board)
                if dist != float('inf'):
                        ratio = diamond.properties.points / dist #
Menghitung ratio
                    if ratio > best_ratio:
                        best_ratio = ratio
                        best_target = diamond
                        best_distance = dist

        if best_target is None:
              target = board_bot.properties.base # Logika untuk bot
kembali pada base
        else:
             target = best_target.position # Logika untuk pergerakan
bot menuju diamond terbaik

        dx, dy = 0, 0
        if my_pos.x < target.x:
            dx = 1 # Pergerakan ke kakan
        elif my_pos.x > target.x:
            dx = -1 # Pergerakan ke kiri
        elif my_pos.y < target.y:
            dy = 1 # Pergerakan ke bawah
        elif my_pos.y > target.y:
            dy = -1 # Pergerakan ke atas

        return dx, dy # Mengembalikan arah gerak
```

## 2. Penjelasan Alur Program

Program bot MpokZenBot pada permainan ini dibangun dengan memanfaatkan dua algoritma utama, yaitu Breadth-First Search (BFS) untuk pencarian jalur terpendek dan Greedy Algorithm yang menggunakan rasio poin terhadap jarak sebagai kriteria pemilihan diamond. Berikut penjelasan rinci dari alur program:

1. Inisialisasi Bot dan Struktur Data
    - Bot didefinisikan dalam class MpokZenBot yang diturunkan dari BaseLogic.
    - Dua struktur data utama yang digunakan adalah:
        - Self.diamonds: menyimpan semua objek diamond yang ditemukan di peta.
        - self.obstacles: menyimpan posisi semua objek penghalang (seperti dinding dan bot lain).
2. Fungsi get_objects
    - Fungsi ini memindai seluruh objek pada peta.
    - Jika objek bertipe Diamond, maka posisinya ditambahkan ke dalam daftar self.diamonds.
    - Jika objek bertipe Bot atau Wall, maka posisinya dimasukkan ke self.obstacles.
    - Tujuannya adalah memisahkan objek yang dapat diambil dan yang harus dihindari.
3. Algoritma Breadth-First Search (BFS)
    - Digunakan dalam fungsi bfs() untuk menghitung jarak terpendek dari posisi bot ke target diamond.
    - BFS menjelajahi peta secara menyeluruh menggunakan struktur antrian deque.
    - Setiap simpul yang dikunjungi akan ditandai agar tidak dikunjungi kembali.
    - Fungsi ini menghindari posisi yang terdapat dalam self.obstacles.
    - Jika diamond dapat dicapai, maka dikembalikan jaraknya. Jika tidak, maka hasilnya infinity.
4. Strategi Greedy dan Rasio Poin/Jarak
    - Dalam fungsi next_move(), bot akan:
        - Mengambil informasi kondisi tas: posisi, jumlah diamond, dan kapasitas.

    - Menghitung sisa ruang tas untuk menentukan apakah diamond dapat diambil.
- Untuk setiap diamond:
    - Dicek apakah diamond memiliki poin dan cukup ruang di tas.
    - Hitung jarak ke diamond menggunakan BFS.
    - Hitung rasio: poin / jarak.
    - Jika rasio lebih tinggi dari sebelumnya, simpan sebagai target terbaik.

5. Pengambilan Keputusan Arah Gerak
    - Jika tidak ada target yang valid, bot akan kembali ke base.
    - Jika ada target terbaik, bot akan bergerak ke arah diamond tersebut.
    - Pergerakan hanya dilakukan satu langkah per giliran, dengan arah (kanan, kiri, atas, bawah) tergantung posisi relatif bot terhadap target.
    - Arah gerak dikembalikan dalam bentuk dx dan dy.

**4.2 Struktur Data yang Digunakan**

Dalam implementasi bot MpokZenBot, digunakan beberapa struktur data utama untuk mendukung proses pencarian diamond dan navigasi peta. Adapun struktur data yang digunakan antara lain:

1. List (self.diamonds)

   Digunakan untuk menyimpan semua objek diamond yang terdapat pada peta. List ini akan diisi ulang setiap kali fungsi get_objects() dipanggil agar mencerminkan kondisi terbaru dari peta permainan.

   ```
   self.diamonds = []
   ```

   Struktur list memungkinkan penyimpanan berurutan dan akses iteratif terhadap setiap diamond untuk proses pemilihan berdasarkan rasio poin dan jarak.

2. Set (self.obstacles)

   Digunakan untuk menyimpan koordinat dari semua rintangan pada peta, seperti tembok dan bot lain. Dengan menggunakan set, proses pengecekan apakah suatu posisi merupakan rintangan menjadi lebih efisien (kompleksitas O(1)).

   ```
   self.obstacles = set()
   ```

Set ini digunakan dalam algoritma BFS untuk menghindari posisi yang tidak bisa dilewati.

3. Queue (collections.deque)

   Digunakan dalam algoritma Breadth-First Search (BFS) untuk menyimpan posisi-posisi yang akan dijelajahi. deque dipilih karena mendukung operasi append dan pop dari kedua ujung dengan kompleksitas O(1).

   ```
   queue = deque([(start.x, start.y, 0)])
   ```

   Struktur ini memungkinkan proses penelusuran tingkat demi tingkat pada peta secara efisien.

4. Tuple (Koordinat Posisi)

   Posisi-posisi di peta direpresentasikan dalam bentuk tuple (x, y). Tuple digunakan untuk menyimpan dan membandingkan posisi karena bersifat immutable dan dapat dijadikan elemen dalam set.

   Contoh penggunaannya:

   ```
   visited.add((x, y))
   ```

5. Objek Kelas (GameObject, Position)

   Digunakan untuk menangani informasi posisi dan properti objek dalam peta permainan:

   - GameObject menyimpan informasi seperti tipe objek, posisi, dan properti (misalnya poin diamond).
   - Position menyimpan koordinat posisi dalam bentuk x dan y.

### 4.3 Pengujian Program

**1. Skenario Pengujian**

Pengujian dilakukan untuk mengevaluasi kinerja bot MpokZenBot dalam berbagai kondisi peta permainan. Tujuan pengujian adalah memastikan bahwa bot dapat:

1. Menghindari rintangan seperti tembok dan bot lain.
2. Memilih diamond yang paling efisien berdasarkan rasio nilai terhadap jarak.
3. Kembali ke base saat tas sudah penuh atau tidak ada diamond yang sesuai kapasitas.

Skenario yang diuji antara lain:

| Skenario | Deskripsi Skenario | Diharapkan |
|---|---|---|
| 1 | Diamond berada dekat dan tidak terhalang rintangan | Bot bergerak langsung ke diamond |
| 2 | Diamond bernilai besar tetapi jauh | Bot memilih diamond dengan rasio terbaik |
| 3 | Tidak ada diamond yang muat di tas | Bot kembali ke base |
| 4 | Rintangan menghalangi jalur langsung ke diamond | Bot menggunakan jalur alternatif (via BFS) |
| 5 | Diamond lebih dari satu, dengan berbagai nilai dan jarak | Bot memilih berdasarkan rasio terbaik (greedy) |

## 2. Hasil Pengujian dan Analisis

Berikut adalah hasil dari beberapa pengujian yang dilakukan:

| Skenario | Hasil Bot | Status |
|---|---|---|
| 1 | Bot berhasil menuju diamond terdekat secara langsung | Sesuai |
| 2 | Bot lebih memilih diamond terdekat dengan rasio lebih baik | Sesuai |
| 3 | Bot langsung kembali ke base karena tidak ada diamond sesuai | Sesuai |
| 4 | Bot menghindari rintangan dan tetap mencapai diamond | Sesuai |
| 5 | Bot menghitung rasio semua diamond dan memilih yang terbaik | Sesuai |

# BAB V
# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, dapat disimpulkan bahwa:

1. Bot MpokZenBot berhasil menggunakan algoritma greedy untuk menentukan langkah terbaik dalam pengambilan diamond berdasarkan perbandingan antara nilai diamond dan jaraknya (rasio value/distance).
2. Algoritma Breadth-First Search (BFS) yang digunakan dapat menemukan jarak terpendek dari posisi bot ke target (diamond atau base) dengan mempertimbangkan rintangan di peta.
3. Bot mampu menghindari rintangan dan memilih jalur alternatif yang tetap efisien.
4. Logika bot mampu memprioritaskan diamond sesuai kapasitas tas dan kembali ke base apabila tidak ada diamond yang sesuai atau kapasitas sudah penuh.

## 5.2 Saran

Untuk pengembangan lebih lanjut, berikut beberapa saran yang dapat diterapkan:

1. Peningkatan strategi pengambilan keputusan, seperti mempertimbangkan waktu permainan atau posisi bot lawan untuk strategi yang lebih dinamis.
2. Pengembangan memori bot agar dapat menyimpan informasi peta yang sudah pernah dijelajahi untuk efisiensi langkah di masa mendatang.
3. Penambahan algoritma penghindaran dinamis terhadap bot lawan agar tidak hanya menghindari posisi diam, tetapi juga memprediksi gerakan lawan.

# LAMPIRAN

## A. Repository Github (link)

# DAFTAR PUSTAKA


[1] R. Munir, *Strategi Algoritma*, Institut Teknologi Bandung, 2024. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/stmik.htm

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[3] E. Horowitz, S. Sahni, and S. Anderson-Freed, *Fundamentals of Data Structures in C*, 2nd ed. Silicon Press, 2008.

[4] M. A. Weiss, *Data Structures and Algorithm Analysis in C++*, 4th ed. Boston: Pearson, 2014.

[5] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Upper Saddle River, NJ: Pearson, 2020.

[6] J. Kleinberg dan É. Tardos, *Algorithm Design*, 1st ed. Boston: Pearson, 2006.

[7] S. Dasgupta, C. H. Papadimitriou, dan U. V. Vazirani, *Algorithms*, New York: McGraw-Hill, 2008.