# IMPLEMENTASI ALGORITMA GREED by DISTANCE 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 3 (Makam Nazarick)**

| | |
|---|---|
| Varasina Farmadani | 123140107 |
| Michael Mathew | 123140101 |
| George Haansraj | 123140001 |

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

**PROGRAM STUDI TEKNIK INFORMATIKA**
**FAKULTAS TEKNOLOGI INDUSTRI**
**INSTITUT TEKNOLOGI SUMATERA**

**2025**

# DAFTAR ISI

# BAB I
# DESKRPISI TUGAS

Pada tugas besar Strategi Algoritma ini, mahasiswa diminta untuk membuat bot yang nantinya akan dipertandingkan satu sama lain melalui pertandingan “Diamonds”. Dalam pembuatan bot, mahasiswa diharuskan untuk menggunakan **strategi *greedy***.

Diamonds merupakan suatu *programming challenge* yang mempertandingkan suatu bot dengan bot lain. Tujuan dari bot ini adalah mengumpulkan *diamond* sebanyak-banyaknya. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya.

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 diimplementasikan dengan algoritma *greedy* untuk bot)
   c. Program utama (*main*) dan utilitas lainnya

Komponen-komponen dari permainan Diamonds, antara lain:

1. Diamonds
   Untuk memenangkan pertandingan , bot yang dibuat harus dapat mengumpulkan *diamond* sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis *diamond* yaitu *diamond* biru dan *diamond* merah. *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, bot akan diberi tugas untuk mengumpulkan diamond dan *base* digunakan untuk menyimpan *diamond* yang sedang dibawa oleh bot. Tiap bot dalam game ini akan memiliki *base*-nya masing-masing. Apabila *diamond* disimpan ke *base, score* bot akan bertambah senilai *diamond* yang dibawa dan *inventory*.

5. Inventory

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

# BAB II
# LANDASAN TEORI

## 2.1. Dasar Teori

Algoritma adalah suatu langkah atau metode yang telah direncanakan secara matang sehingga terurut dan terorganisir dengan baik dan biasanya digunakan untuk memecahkan suatu masalah dengan memberikan suatu instruksi sehingga menjadi suatu tindakan. Sedangkan dalam Kamus Besar Bahasa Indonesia (KBBI), algoritma adalah suatu prosedur sistematis untuk menyelesaikan masalah matematika dalam langkah-langkah terbatas atau urutan pengambilan keputusan yang logis untuk memecahkan masalah tersebut. Dari pengertian tersebut dapat dikatakan bahwa algoritma ini digunakan untuk menyelesaikan atau menyelesaikan suatu masalah dengan langkah-langkah logis yang terurut.

Dalam dunia pemecahan masalah komputasi, strategi algoritma *greedy* merupakan sebuah strategi yang mengandalkan intuisi sesaat untuk mencapai tujuan. Pendekatan ini bekerja dengan mengambil langkah-langkah yang, pada setiap momen pengambilan keputusan, tampak sebagai pilihan paling menguntungkan. Ibaratnya, setiap ada kesempatan, pilihan yang paling "menggiurkan" saat itu juga akan diambil, dengan sebuah harapan besar bahwa serangkaian keputusan lokal yang optimal ini akan secara ajaib menuntun pada solusi keseluruhan yang juga optimal.

### 2.1.1. Cara Implementasi Program

Implementasi greedy pada program bot kami sini adalah bot akan menentukan langkah apa yang “terbaik” selanjutnya untuk dieksekusi. Dalam pengambilan keputusan, terdapat tiga prioritas utama:

1. Prioritas 1 - Kembali ke base jika inventory penuh

   Ini adalah keputusan greedy yang paling prioritas. Jika bot membawa berlian sebanyak atau lebih dari kapasitas inventory-nya, ia segera dan tanpa ragu memutuskan untuk kembali ke base. Tidak ada perhitungan jarak ke berlian lain atau risiko musuh yang dipertimbangkan; tujuannya adalah menyetor berlian secepatnya. Ini adalah greedy karena memaksimalkan perolehan poin instan.

2. Prioritas 2 - greed by distance (kejar berlian dengan jarak terdekat dari posisi saat ini)

Jika inventory tidak penuh, bot akan secara greedy mengejar berlian terdekat yang tersedia di board. Metrik _manhattan (Manhattan distance) digunakan untuk menemukan yang terdekat. Ini adalah definisi greedy klasik: pilih yang paling mudah dijangkau saat ini, tanpa memikirkan nilai berlian, bahaya di sekitar berlian, atau apakah ada berlian yang lebih strategis tetapi sedikit lebih jauh.

3. Prioritas 3 - Kembali ke base jika sudah tidak ada berlian di board

   Jika tidak ada berlian di board yang bisa diambil (atau inventory sudah penuh, yang ditangani oleh prioritas 1), bot secara greedy memutuskan untuk kembali ke base. Ini mungkin untuk reset atau menunggu berlian baru muncul.

### 2.1.2. Menjalankan Bot Program

Program bot "Nazarick" dieksekusi sebagai skrip Python utama yang berisi seluruh strategi algoritma greedy dan logika pergerakan bot. Pada setiap langkah permainan, game engine akan memanggil bot untuk menentukan pergerakan selanjutnya. Bot memperoleh seluruh informasi status permainan, termasuk peta dan posisi objek (berlian, base, musuh, dll.), melalui API yang disediakan oleh game engine. Data ini kemudian diproses oleh logika greedy bot untuk membuat keputusan optimal secara lokal, yang selanjutnya dieksekusi oleh game engine. Game engine dan bot dijalankan dalam kontainer Docker untuk menjamin konsistensi lingkungan eksekusi.

# BAB III
# APLIKASI STARTEGI GREEDY

## 3.1. Proses Mapping

### 3.1.1 Identifikasi Masalah

Bot diharuskan mengoleksi skor sebanyak banyaknya, diamond yang memiliki poin ada 2 tipe yaitu : Diamond merah, dan Diamond biru. Diamond merah bernilai 2, Diamond biru bernilai 1. Bot memiliki batasan *inventory*, yaitu maksimal 5, bot dapat mempertimbangakan untuk membawa diamond secukupnya, baik dari mengambil pada board atau men-*tackle* lawan. Di map ada item diamond merah, digunakan untuk mengacak posisi dan jumlah diamond pada tiles. Bot dibatasi atau diharuskan menggunakan algoritma Greedy.

### 3.1.2 Pendekatan Greedy

Kita menggunakan Greedy by Distance (dengan batasan inventory) yaitu mengunjungi diamond terdekat namun kita batasi juga jumlah poin yang di bawa, dimana default kita batasi 3 poin, mengingat waktu yang terbatas dan resiko di *tackel* lawan jika berada di luar base.

## 3.2. Eksplorasi Alternatif Solusi greedy

### 3.2.1 Greedy By Value/Points

Greedy ini mendahulukan nilai/poin dari diamond, greedy tipe ini tidak kami pertimbangkan sebagai opsi optimal mengingat sangat terbatasnya diamond merah di dalam map, lalu tidak ada jaminan safe juga meski diamond merah jumlahnya lebih sedikit dibandingkan yang biru.

### 3.2.2 Greedy By Safety/Risk

Greedy ini membatasi jarak minimum antar bot, greedy ini mungkin saja dapat menjadi solusi paling optimal karena ia mengambil diamond sembari menjaga jarak aman, namun greedy ini bisa saja menjadi error jika di apit oleh banyak bot lain. Beberapa solusi atas kemungkinan *error*nya adalah

1. Melakukan gerakan *random* (namun ini akan merusak logic safety nya)
2. Menambahkan fungsi heuristic (dimana melanggar aturan yaitu Menggunakan Greedy)

## 3.3. Analisis Efisiensi dan Efektivitas Solusi Greedy

### 3.3.1 Efisiensi Algoritma (Kompleksitas Waktu)

1. Pencarian Target (GreedyTargeting.find_target)
    - Pemeriksaan kondisi inventory penuh: O(1).
    - Pencarian diamond terdekat: O(N)
        - Iterasi dilakukan sebanyak jumlah diamond: O(N)
        - Perhitungan jarak Manhattan: O(1)
    - Kesimpulan Pencarian Target = O(N)
2. Penentuan Gerakan (GreedyMovement.move_towards): O(1)
3. Deteksi Terjebak (NazarickNPC._is_stuck): O(1)

### 3.3.2 Efektifitas Strategi

- Strenght
    - Responsif terhadap peluang Terdekat
        - Bot akan dengan cepat memanfaatkan diamond yang berada di dekatnya
    - Pengurangan Resiko
        - Dengan adanya batasan diamond yang dibawa (threshold default = 3) sebelum inventory penuh, mengurangi risiko kehilangan banyak diamond jika di-*tackle*
    - Fokus pada Objektif Utama
        - Memprioritaskan diamond/poin, meminimalkan resiko, kembali ke base.
- Weakness
    - Mengabaikan value
        - Bot mungkin memilih diamond biru yang terdekat dibandingkan diamond merah yang lebih jauh.
    - Tidak adaptif
        - Bot tidak mempertimbangkan keberadaan musuh, posisi teleporter, ataupun tombol reset.
    - Potensi pergerakan tidak efisien jangka panjang
        - Diamond terdekat satu persatu mungkin bukan rute terbaik untuk mendapatkan poin yang lebih banyak, karena kepadatan mungkin berbeda beda.

## 3.4. Strategi Greedy yang Dipilih

Greedy By Distance (with inventory limit), dengan detail strategi seperti berikut:

- Prioritas 1: Kembali ke Base jika mencapai Threshold Diamond yang dibawa.
- Prioritas 2: Menuju Diamond terdekat
- Prioritas 3: Kembali Ke Base (Fallback)

# BAB IV
# IMPLEMENTASI DAN PENGUJIAN

## 4.1. Implementasi Algoritma Greedy

### 4.1.1. Pseudocode

```
CLASS GreedyBotConfig:
  RETURN_THRESHOLD_DIAMONDS = 3
  DEFAULT_INVENTORY_SIZE = 5
  STUCK_THRESHOLD = 3

CLASS GreedyMovement:
  FUNCTION move_towards(current_position, target_position, board):
    dx = target_position.x - current_position.x
    dy = target_position.y - current_position.y

    primary_move = (0, 0)
    secondary_move = (0, 0)

    IF ABS(dx) > ABS(dy):
      primary_move = (1 IF dx > 0 ELSE -1, 0)
      IF dy != 0:
        secondary_move = (0, 1 IF dy > 0 ELSE -1)
    ELSE IF ABS(dy) > ABS(dx):
      primary_move = (0, 1 IF dy > 0 ELSE -1)
      IF dx != 0:
        secondary_move = (1 IF dx > 0 ELSE -1, 0)
    ELSE IF dx != 0:
      primary_move = (1 IF dx > 0 ELSE -1, 0)
      secondary_move = (0, 1 IF dy > 0 ELSE -1)
    ELSE:
      RETURN (0, 0)

    IF primary_move != (0, 0) AND board.is_valid_move(current_position, primary_move.x, primary_move.y):
      RETURN primary_move

    IF secondary_move != (0, 0) AND board.is_valid_move(current_position, secondary_move.x,
secondary_move.y):
      RETURN secondary_move

    moves_fallback = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    SHUFFLE moves_fallback
    FOR each move in moves_fallback:
      IF move != primary_move AND move != secondary_move:
        IF board.is_valid_move(current_position, move.x, move.y):
          RETURN move
```

```
    RETURN (0, 0)

  FUNCTION get_random_move(current_position, board):
    moves = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    SHUFFLE moves
    FOR each move in moves:
      IF board.is_valid_move(current_position, move.x, move.y):
        RETURN move
    RETURN (0, 0)

CLASS GreedyTargeting:
  FUNCTION _manhattan(position_a, position_b):
    RETURN ABS(position_a.x - position_b.x) + ABS(position_a.y - position_b.y)

  FUNCTION find_target(board_bot, board):
    props = board_bot.properties
    current_pos = board_bot.position

    IF NOT props OR NOT props.base:
      RETURN current_pos

    diamonds_carried = props.diamonds IF props.diamonds IS NOT None ELSE 0
    inventory_size = props.inventory_size IF props.inventory_size IS NOT None ELSE
GreedyBotConfig.DEFAULT_INVENTORY_SIZE

    is_inventory_full = diamonds_carried >= inventory_size
    has_enough_diamonds_threshold_met = diamonds_carried >= GreedyBotConfig.RETURN_THRESHOLD_DIAMONDS

    IF is_inventory_full OR has_enough_diamonds_threshold_met:
      RETURN props.base

    IF board.diamonds EXISTS:
      valid_diamonds = [d FOR d IN board.diamonds IF d.position EXISTS]
      IF valid_diamonds EXISTS:
        closest_diamond = diamond IN valid_diamonds WITH MINIMUM _manhattan(current_pos, diamond.position)
        RETURN closest_diamond.position

    RETURN props.base
```

```
CLASS NazarickNPC INHERITS BaseLogic:
  CONSTRUCTOR:
    self.movement = NEW GreedyMovement()
    self.targeting = NEW GreedyTargeting()
    self.last_position = None
    self.stuck_counter = 0

  FUNCTION next_move(board_bot, board):
    current_pos = board_bot.position

    IF self.last_position EXISTS AND self._positions_equal(self.last_position, current_pos):
      self.stuck_counter = self.stuck_counter + 1
    ELSE:
      self.stuck_counter = 0
    self.last_position = current_pos (copy)

    IF self.stuck_counter >= GreedyBotConfig.STUCK_THRESHOLD:
      RETURN self.movement.get_random_move(current_pos, board)

    target_pos = self.targeting.find_target(board_bot, board)
    RETURN self.movement.move_towards(current_pos, target_pos, board)

  FUNCTION _positions_equal(position_a, position_b):
    RETURN position_a.x == position_b.x AND position_a.y == position_b.y
```

### 4.1.2. Penjelasan Alur Program

Program bot NazarickNPC dirancang untuk bermain secara otomatis menggunakan pendekatan greedy (serakah), yaitu selalu bergerak menuju target terdekat yang paling menguntungkan (misalnya diamond atau base). Algoritma ini

memanfaatkan logika sederhana namun efektif untuk pengambilan keputusan berdasarkan kondisi saat ini. Berikut adalah penjelasan alurnya:

1. Konfigurasi Bot (GreedyBoyConfig)

   Kelas ini menyimpan nilai-nilai konfigurasi dasar seperti:

   - RETURN_THRESHOLD_DIAMONDS

     jumlah diamond minimum sebelum kembali ke base.

   - DEFAULT_INVENTORY_SIZE

     kapasitas default jika data tidak tersedia.

   - STUCK_THRESHOLD

     jumlah giliran tanpa bergerak sebagai batas dianggap "stuck".

2. Penentuan Target (GreedyTargeting)

   Kelas ini bertanggung jawab untuk menentukan tujuan utama bot berdasarkan kondisi saat ini:

   - Jika inventory penuh atau jumlah diamond sudah cukup (≥ threshold), maka target adalah base.
   - Jika masih bisa membawa diamond:
     1. Akan mencari diamond terdekat menggunakan perhitungan Manhattan Distance.
     2. Jika tidak ada diamond yang terdeteksi, maka fallback target tetap base.

3. Pergerakan Menuju Target (GreedyMovement)

   Kelas ini mengatur bagaimana bot bergerak menuju posisi target dengan strategi sebagai berikut:

   - Menghitung selisih koordinat dx dan dy.
   - Menentukan arah gerakan utama (horizontal atau vertikal) berdasarkan nilai selisih terbesar.
   - Jika gerakan utama tidak valid (misalnya terhalang), akan mencoba gerakan sekunder.
   - Jika kedua arah tidak valid, maka akan mencoba arah cadangan (fallback) yang diacak.
   - Jika semua opsi gagal, bot akan tetap di tempat ((0, 0)).

   Juga terdapat fungsi get_random_move yang akan memilih langkah secara acak jika bot dianggap stuck.

4. Logika Utama Bot (NazarickNPC)

   Kelas utama ini menggabungkan semua komponen:

   - Menyimpan posisi terakhir dan menghitung apakah bot mengalami kemandekan (stuck).
   - Jika bot tidak berpindah posisi dalam beberapa giliran (≥ threshold), maka akan bergerak secara acak untuk membebaskan diri.
   - Jika tidak stuck, bot akan:
     1. Menentukan target menggunakan GreedyTargeting.
     2. Bergerak ke arah target menggunakan GreedyMovement.

Alur Eksekusi next_move() secara Rinci:

1. Bot membaca posisi saat ini.
2. Mengecek apakah bot berpindah dari posisi sebelumnya:
   - Jika tidak, tingkat stuck_counter meningkat.
   - Jika iya, stuck_counter direset.
3. Jika sudah stuck, pilih gerakan acak.
4. Jika tidak stuck:
   - Tentukan target berdasarkan kondisi inventory.
   - Lakukan gerakan menuju target.
5. Kembalikan nilai arah gerakan yang dipilih.

Keunggulan Pendekatan Greedy Ini:

1. Cepat dan sederhana.
2. Responsif terhadap perubahan kondisi (misal: kehabisan diamond, inventory penuh).
3. Menghindari posisi terjebak (dengan deteksi stuck dan random movement).

## 4.2. Struktur Data yang Digunakan

Program bot ini menggunakan beberapa struktur data utama, sebagian besar didefinisikan dalam file game/models.py menggunakan dataclasses untuk representasi objek yang jelas dan terstruktur, serta tipe data dasar Python:

1. Dataclasses (@dataclass):

- Bot: Merepresentasikan informasi dasar bot seperti nama, email, dan ID.
- Position: Merepresentasikan koordinat X dan Y pada papan permainan. Ini adalah dasar untuk posisi objek dan markas.
- Base: Mewarisi dari Position, merepresentasikan posisi markas bot.
- Properties: Menyimpan berbagai properti yang terkait dengan GameObject, seperti poin, ID pasangan, berlian yang dikumpulkan, skor, nama bot, ukuran inventaris, kemampuan tackling, dan waktu yang tersisa.
- GameObject: Merepresentasikan objek apa pun di papan permainan (misalnya, bot, berlian). Ini memiliki ID, posisi, tipe, dan properti opsional.
- Config: Digunakan di dalam Feature untuk mengkonfigurasi aspek-aspek tertentu dari papan permainan.
- Feature: Merepresentasikan fitur spesifik pada papan permainan (misalnya, jenis generasi berlian) dan konfigurasinya.
- Board: Struktur data paling komprehensif yang merepresentasikan keadaan seluruh papan permainan, termasuk lebar, tinggi, fitur, penundaan minimum antar gerakan, dan daftar objek permainan. Ini juga memiliki properti untuk menyaring bots dan diamonds dari game_objects.

2. List (List):

   Digunakan secara ekstensif untuk koleksi objek, seperti:

   - features dalam Board.
   - game_objects dalam Board.
   - Properti @property bots dan diamonds dalam Board mengembalikan daftar GameObject.
   - directions dalam RandomLogic (list of tuples).
   - moves dan moves_fallback dalam GreedyMovement (list of tuples).

3. Dictionary (dict):

   Digunakan dalam main.py untuk memetakan nama logika string ke kelas logika bot yang sesuai (misalnya, "Random": RandomLogic, "Nazarick": NazarickNPC).

4. Tuple (Tuple):

Digunakan untuk merepresentasikan koordinat pergerakan (delta_x, delta_y) atau arah pergerakan (1, 0), (0, 1), dll. Ini sangat umum dalam fungsi next_move dan pergerakan.

5. Tipe Data Dasar Python:
    - Integer (int): Untuk ID, koordinat (x, y), lebar, tinggi, jumlah berlian, skor, dan penghitung (stuck_counter).
    - String (str): Untuk nama, email, password, ID, tipe objek ("BotGameObject", "DiamondGameObject"), dan arah pergerakan ("NORTH", "EAST").
    - Boolean (bool): Untuk nilai benar/salah, misalnya dalam can_tackle atau hasil dari is_valid_move.
6. Optional dan Union dari typing module:
    - Digunakan untuk memberikan petunjuk tipe (type hints) yang menunjukkan bahwa suatu variabel atau parameter dapat berupa tipe tertentu atau None (Optional), atau dapat berupa salah satu dari beberapa tipe (Union). Ini meningkatkan keterbacaan dan pemeliharaan kode.

Struktur data ini bekerja sama untuk secara efektif merepresentasikan keadaan permainan, informasi bot, dan logika keputusan yang diperlukan untuk fungsionalitas bot. Program bot "Nazarick" menggunakan berbagai struktur data Python untuk merepresentasikan entitas game dan mengelola logikanya. Struktur data utama yang digunakan meliputi:

1. Dataclasses (@dataclass):

    Digunakan secara ekstensif di game/models.py untuk mendefinisikan struktur data yang bersih dan ringkas untuk objek-objek game. Dataclass secara otomatis menghasilkan metode seperti __init__, __repr__, dan __eq__.

    Contohnya:

    - Bot: Merepresentasikan bot dengan name, email, dan id.
    - Position: Merepresentasikan koordinat X dan Y pada papan.
    - Base: Mewarisi dari Position dan merepresentasikan posisi markas.
    - Properties: Menyimpan berbagai properti bot atau objek lain seperti points, diamonds, score, inventory_size, dll.. Properti ini sering bersifat opsional (Optional).
    - GameObject: Merepresentasikan objek umum di papan dengan id, position, type, dan properties.

- Config: Mengandung konfigurasi untuk fitur papan seperti generation_ratio, seconds, pairs, dll..
- Feature: Merepresentasikan fitur papan dengan name dan config opsional.
- Board: Merepresentasikan keadaan papan permainan, termasuk id, width, height, daftar features, minimum_delay_between_moves, dan daftar game_objects. Kelas ini juga memiliki properti @property untuk mengakses daftar bots dan diamonds dengan mudah dari game_objects.

2. List (List):
    - Digunakan untuk menyimpan koleksi objek, seperti daftar features dan game_objects di dalam kelas Board.
    - Digunakan juga di nazarick_movement.py untuk mendefinisikan daftar arah pergerakan yang mungkin [(1, 0), (0, 1), (-1, 0), (0, -1)].
    - Digunakan untuk menyimpan daftar berlian yang valid (valid_diamonds) di nazarick_targeting.py.
3. Tuple (Tuple):

    Digunakan untuk merepresentasikan pergerakan (delta_x, delta_y) sebagai pasangan integer, misalnya (1, 0) untuk bergerak ke timur. Fungsi next_move di BaseLogic dan kelas turunan seperti NazarickNPC mengembalikan tuple (delta_x, delta_y).

4. Dictionary (dict):

    Digunakan dalam main.py untuk memetakan nama logika string ke kelas logika bot yang sesuai dalam variabel CONTROLLERS. Ini memungkinkan program untuk memilih logika bot secara dinamis berdasarkan argumen baris perintah. API response yang diterima dari server juga dalam format dictionary sebelum di-deserialisasi menjadi objek dataclass.

5. None (digunakan dengan Optional):

    Digunakan secara luas dengan *type hinting* Optional dari modul typing. Ini menunjukkan bahwa suatu variabel atau atribut dapat berupa tipe data tertentu atau None. Contohnya terlihat pada banyak atribut di kelas Properties. Ini penting untuk menangani kasus di mana data mungkin tidak selalu tersedia dari API.

## 4.3. Pengujian Program

Pengujian bot "Nazarick" dilakukan untuk memverifikasi fungsionalitas dan kinerja dasar bot dalam lingkungan permainan Diamonds. Ini melibatkan observasi perilaku bot saat berinteraksi dengan papan permainan, berlian, dan markas, serta kemampuannya untuk mengatasi situasi tertentu.

Langkah-langkah Pengujian:

1. Siapkan Lingkungan:
    - Pastikan device memiliki Python yang sudah terinstal.
    - Instal semua dependensi yang diperlukan dari requirements.txt dengan menjalankan pip install -r requirements.txt di terminal dari direktori src.
    - Pastikan game engine Diamonds berjalan.
2. Jalankan Bot:

    Untuk satu bot: jalankan satu instance bot "Nazarick" menggunakan perintah di bash:

    ```
    python main.py --logic Nazarick --email=your_email@example.com --name=your_name
    --password=your_password --team etimo
    ```

    Ganti your_email@example.com, your_name, dan your_password dengan detail yang diinginkan.

    Untuk beberapa bot: jalankan banyak bot secara bersamaan menggunakan skrip run-bots.sh (untuk Linux/macOS) atau run-bots.bat (untuk Windows). Pastikan setiap bot memiliki email dan nama yang unik saat menjalankan beberapa bot.

3. Observasi:
    - Amati perilaku bot secara langsung di antarmuka game Diamonds (jika tersedia) atau melalui output konsol yang diberikan oleh skrip main.py.
    - Catat apakah perilaku bot sesuai dengan hasil yang diharapkan untuk setiap skenario yang diuji.
4. Analisis Konfigurasi:

    Perhatikan parameter konfigurasi bot yang digunakan, seperti RETURN_THRESHOLD_DIAMONDS, DEFAULT_INVENTORY_SIZE, dan STUCK_THRESHOLD yang didefinisikan dalam nazarick_config.py. Pengaturan ini secara langsung memengaruhi perilaku bot.

Tujuan Pengujian:

Tujuan utama dari pengujian ini adalah untuk memverifikasi fungsionalitas inti dari bot "Nazarick", meliputi:

- Akuisisi Target: Memastikan bot dapat mengidentifikasi berlian terdekat atau markas sebagai target pergerakannya.
- Navigasi: Memastikan bot dapat bergerak secara valid di papan permainan menuju targetnya, serta menangani batasan papan.
- Manajemen Inventaris: Memverifikasi bahwa bot kembali ke markas saat memiliki jumlah berlian yang cukup atau inventaris penuh.
- Penanganan Kondisi Abnormal: Memastikan bot dapat pulih dari kondisi "terjebak" di posisi yang sama.
- Konkurensi: Memastikan bahwa beberapa instance bot dapat berjalan dan berinteraksi dengan game engine secara bersamaan tanpa masalah.

### 4.3.1. Skenario Program

| Skenario Pengujian | Tujuan Pengujian | Prosedur Pengujian | Hasil yang Diharapkan | Hasil Aktual |
|---|---|---|---|---|
| Pengambilan Berlian Normal | Memverifikasi kemampuan bot untuk menemukan dan mengambil berlian terdekat. | Mulai game engine. Jalankan bot "Nazarick" (python main.py --logic Nazarick --email=test@email.com --name=stima --password=123456 --team etimo). Amati pergerakan bot menuju berlian yang muncul di papan. | Bot harus bergerak secara efisien menuju berlian terdekat yang terlihat, dihitung menggunakan jarak Manhattan (abs(a.x - b.x) + abs(a.y - b.y)). | Berhasil |
| Kembali ke Markas dengan Berlian Cukup | Memverifikasi kemampuan bot untuk kembali ke markas saat telah mengumpulkan jumlah berlian yang memadai. | Mulai game engine. Jalankan bot "Nazarick". Amati pergerakan bot setelah mengumpulkan 3 berlian atau lebih (sesuai RETURN_THRESHOLD_DIAMONDS). | Bot harus mengubah targetnya dari berlian menjadi markas (props.base) dan bergerak kembali ke posisi markasnya. | Berhasil |
| Penanganan Kondisi "Terjebak" | Memverifikasi bahwa bot dapat pulih dari kondisi terjebak (misalnya, terhalang oleh objek lain atau dinding). | Mulai game engine. Jalankan bot "Nazarick". Coba simulasi kondisi di mana bot terhalang pergerakannya secara terus-menerus. Amati perilaku bot setelah berada di posisi yang sama selama 3 langkah atau lebih (sesuai STUCK_THRESHOLD). | Bot harus melakukan pergerakan acak (get_random_move) untuk mencoba keluar dari posisi terjebak. | Berhasil |

| | | | | |
|---|---|---|---|---|
| Pergerakan Fallback | Memverifikasi bahwa bot dapat memilih pergerakan alternatif jika pergerakan primernya tidak valid. | Mulai game engine. Jalankan bot "Nazarick". Amati pergerakan bot di dekat sudut atau tepi papan di mana beberapa arah pergerakan mungkin tidak valid. | Bot harus mampu memilih pergerakan valid berikutnya yang tersedia dari daftar moves_fallback jika pergerakan utama dan sekunder tidak valid. | Berhasil |
| Multi-bot Concurrency | Memastikan beberapa instance bot dapat berjalan bersamaan tanpa konflik identitas atau pergerakan dasar. | Mulai game engine. Jalankan skrip run-bots.sh. Pastikan setiap bot memiliki email dan nama yang unik. Amati pergerakan masing-masing bot secara simultan di papan. | Semua bot harus berhasil terdaftar, bergabung dengan papan, dan mulai bermain secara independen. | Berhasil |

### 4.3.2. Hasil Pengujian dan Analisis

Berdasarkan pengujian yang dilakukan, bot "Nazarick" menunjukkan fungsionalitas dasar yang diharapkan untuk strategi *greedy* berdasarkan jarak Manhattan. Bot dapat menemukan targetnya, bergerak menuju target, kembali ke markas, dan menangani situasi terjebak. Kemampuan untuk menjalankan beberapa bot secara bersamaan juga diverifikasi.

Rekomendasi untuk Pengembangan Lanjutan:

- Optimasi Pergerakan: Meskipun pergerakan dasar berfungsi, eksplorasi algoritma pencarian jalur yang lebih canggih (misalnya A*) dapat meningkatkan efisiensi pergerakan dan penanganan rintangan dinamis.
- Tackling/Interaksi Bot: Jika fitur *tackling* diaktifkan di game engine, pertimbangkan untuk mengimplementasikan logika untuk menghindari atau melakukan *tackle* terhadap bot lawan.
- Strategi Berlian Spesifik: Mengembangkan logika yang lebih canggih untuk memilih berlian (misalnya, berdasarkan nilai berlian, posisi bot lain, atau pola *spawn*) dapat meningkatkan skor.

# BAB V
# KESIMPULAN DAN SARAN

## 5.1. Kesimpulan

Dalam mengumpulkan diamond, bot “Nazarick” mengimplementasikan algoritma *greedy* dengan pendekatan berdasarkan jarak. Strategi ini diwujudkan melalui :

- Prioritas target : bot secara *greedy* akan memprioritaskan penyetoran berlian ke *base* jika *inventory* penuh. Ini merupakan keputusan yang sangat *greedy* karena langsung memaksimalkan poin yang diamankan tanpa memikirkan peluang berlian lain.
- Modul pergerakan : selalu memilih langkah tunggal yang paling mendekatkan bot ke target yang dipilih, memprioritaskan sumbu dengan perbedaan jarak terbesar.

## 5.2. Saran

Algoritma ini masih dapat dikembangkan lagi dengan memperkaya strategi *greedy* yang ada untuk meningkatkan potensi performa yang lebih baik dalam permainan “Diamonds”, sebagai contoh :

- Membuat variabel yang menghitung weight diamond dengan beberapa parameter masukan seperti nilai diamond, jarak diamond, lokasi teleport, dll.
- Pemanfaatan teleporter yang dapat meningkatkan mobilisasi atau pergerakan dari bot dalam mengumpulkan diamond.

# LAMPIRAN

Link Repository GitHub : https://github.com/sinavarasina/Tubes1_MakamNazarick

# DAFTAR PUSTAKA

A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd ed, Boston, MA: USA: Pearson Education, Inc, 2012.

V. Chugani, "What is Manhattan Distance? A Deep Dive," DataCamp, 17 July 2024. [Online]. Available: https://www.datacamp.com/tutorial/manhattan-distance. [Accessed 1 June 2025].

S. Jani, "Greedy Algorithms," GeeksforGeeks, 7 April 2025. [Online]. Available: https://www.geeksforgeeks.org/greedy-algorithms/. [Accessed 1 June 2025].

R. Munir, Strategi Algoritma, Bandung: Institut Teknologi Bandung, 2024.