# IMPLEMENTASI ALGORITMA X DALAM PEMECAHAN BOT PERMAINAN DIAMOND

## Tugas Besar

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


**Oleh: Kelompok Velocity**

1. **Khorian Mukhsin** **(123140006)**
2. **Ade Putri Tifani** **(123140011)**
3. **Natasya Felisita Br Ginting** **(123140017)**

**Dosen Pengampu: Winda Yulita, M.Cs.**

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

**2025**

# DAFTAR ISI

# BAB I
# DESKRIPSI TUGAS

## 1.1 Latar Belakang

"Diamonds" adalah sebuah permainan berbasis pemrograman yang menantang para peserta untuk membuat bot yang dapat bersaing dengan bot milik pemain lain. Tujuan utama dari game ini adalah mengumpulkan diamond sebanyak-banyaknya di dalam permainan yang penuh tantangan. Bot yang dibuat harus mampu mengambil keputusan yang tepat dan cepat untuk mengoptimalkan jumlah poin yang didapat.

## 1.2 Komponen Permainan

Permainan ini terdiri dari dua komponen utama:

1. Game Engine:
   - Backend: Mengatur logika permainan dan menyediakan API untuk interaksi antara frontend dan bot.
   - Frontend: Menyediakan tampilan visual dari permainan.

2. Bot Starter Pack:
   - API Client: Untuk berkomunikasi dengan backend.
   - Bot Logic: Bagian yang akan diimplementasikan menggunakan algoritma greedy.
   - Program Utama & Utilitas: Untuk menjalankan dan mendukung bot.

## 1.3 Aturan Permainan:

- Setiap bot memiliki sebuah Base tempat menyimpan diamond.
- Red Diamond bernilai 2 poin, sedangkan Blue Diamond bernilai 1 poin.
- Red Button: Jika diinjak, seluruh diamond akan di-regenerate secara acak di posisi baru, termasuk red button itu sendiri.
- Teleporters: Dua titik yang saling terhubung. Jika bot masuk ke salah satu, bot akan keluar di titik lainnya.
- Bot hanya mendapatkan poin jika berhasil mengembalikan diamond yang dibawa ke Base.
- Permainan berlangsung secara real-time dengan batas waktu gerak untuk setiap bot.

# BAB II

# LANDASAN TEORI

## 2.1 Dasar Teori

Algoritma greedy, atau algoritma serakah, merupakan salah satu metode dalam penyelesaian masalah optimasi yang bekerja dengan cara membangun solusi secara bertahap. Pada setiap langkah, algoritma ini memilih opsi yang tampak terbaik atau paling optimal secara lokal, tanpa mempertimbangkan dampak keputusan tersebut terhadap langkah-langkah berikutnya. Dengan kata lain, algoritma greedy mengambil keputusan berdasarkan keuntungan langsung terbesar yang bisa diperoleh saat itu, dengan harapan bahwa rangkaian keputusan lokal ini akan mengarah pada solusi global yang optimal.

Secara umum, proses kerja algoritma greedy melibatkan tiga tahap utama, yaitu inisialisasi solusi awal, pemilihan komponen terbaik berdasarkan kriteria tertentu, dan validasi keputusan tersebut untuk memastikan bahwa komponen tersebut dapat ditambahkan ke solusi tanpa melanggar batasan atau kendala yang ada. Proses ini dilakukan secara berulang hingga solusi akhir terbentuk atau tidak ada lagi pilihan yang tersedia.

Namun, penting untuk dicatat bahwa algoritma greedy tidak selalu menghasilkan solusi yang optimal untuk semua jenis masalah. Keberhasilan algoritma ini bergantung pada dua sifat penting yang harus dimiliki oleh permasalahan yang diselesaikan, yaitu *greedy choice property* dan *optimal substructure*. *Greedy choice property* menyatakan bahwa pilihan terbaik pada setiap langkah akan menghasilkan solusi optimal secara keseluruhan. Sementara itu, *optimal substructure* berarti bahwa solusi optimal dari suatu masalah dapat dibangun dari solusi optimal dari submasalah-submasalah nya. Apabila kedua sifat tersebut tidak terpenuhi, maka pendekatan greedy dapat memberikan hasil yang sub-optimal dan kurang tepat. Dalam kasus seperti itu, metode lain seperti *dynamic programming* sering kali lebih tepat digunakan.

Kelebihan dari algoritma greedy terletak pada kesederhanaannya dan efisiensinya dalam penggunaan waktu serta memori, karena algoritma ini tidak perlu mengevaluasi seluruh kemungkinan solusi. Namun demikian, kelemahan utamanya adalah tidak adanya jaminan bahwa solusi yang diperoleh merupakan solusi terbaik secara keseluruhan.

Dengan berbagai karakteristik tersebut, algoritma greedy banyak digunakan dalam berbagai aplikasi seperti penyusunan jadwal, pengkodean Huffman, serta algoritma graf seperti Kruskal dan Prim.

Sebagaimana dijelaskan oleh Cormen et al. dalam *Introduction to Algorithms* (Cormen et al., 2009, [1] dan Dasgupta et al. dalam *Algorithms* [2], algoritma greedy merupakan pendekatan yang efektif jika digunakan dalam konteks permasalahan yang tepat.

### 2.1.1 Cara Implementasi Program

Implementasi program ini bertujuan untuk mengembangkan sebuah bot yang mampu bergerak secara otomatis di dalam sebuah papan permainan (board) yang berguna mengumpulkan objek diamond, dan kembali ke base (markas) untuk menyimpan dan menyetorkan hasil yang telah dikumpulkan. Strategi yang digunakan dalam pengambilan keputusan adalah strategi greedy.

- **Inisialisasi Kelas Bot**

  Kelas VelocityBot merupakan turunan dari kelas BaseLogic, yang merupakan abstraksi logika utama dalam permainan. Pada saat inisialisasi, dua atribut utama didefinisikan

```
class VelocityBot (BaseLogic):
    def __init__(self):
        self.goal_position = None
        self.returning_to_base = False
```

  - ➢ Kelas VelocityBot diturunkan dari kelas BaseLogic, yang merupakan kerangka dasar logika pergerakan bot.
  - ➢ goal_position: menyimpan posisi tujuan saat ini.
  - ➢ returning_to_base: flag boolean yang menandai apakah bot sedang dalam perjalanan kembali ke base untuk menyetor diamond.

- **Fungsi Utama: next_move**

  Fungsi ini dieksekusi di setiap langkah permainan untuk menentukan tindakan bot selanjutnya:

```
def next_move(self, board_bot: GameObject, board: Board):
```

  - ➢ Informasi yang dibaca mencakup posisi bot, jumlah diamond yang dikumpulkan, kapasitas maksimum inventori, serta posisi base.
  - ➢ Berdasarkan status returning_to_base, bot akan memutuskan untuk kembali ke base atau melanjutkan pencarian diamond.

- **Proses Kembali ke Base: handle_return_to_base**

  Apabila kapasitas inventory telah mencapai batas maksimum, bot diarahkan kembali ke base:

```
def handle_return_to_base(self, bot_pos, base_pos, board_bot, board):
```

- ➢ Jika posisi bot telah sampai di base, maka proses penyetoran diamond dilakukan (diwakili dengan mengosongkan inventori).
- ➢ Setelah itu, status bot akan diatur untuk kembali berburu diamond.

- **Proses Pengumpulan Diamond: handle_diamond_collection**

  Jika inventori belum penuh, maka bot akan mencari diamond terbaik yang tersedia:

```
def handle_diamond_collection(self, bot_pos, inventory, inventory_limit, board,
board_bot):
```

  - ➢ Jika jumlah diamond telah mencapai batas kapasitas inventori, bot akan beralih ke mode kembali ke base.
  - ➢ Jika masih tersedia ruang, maka fungsi get_best_diamond_position akan menentukan target diamond terbaik yang akan dikejar berdasarkan nilai dan jaraknya.

- **Menentukan Target Diamond: get_best_diamond_position**

  Fungsi ini digunakan untuk memilih diamond paling menguntungkan berdasarkan nilai dan jaraknya dari posisi bot:

```
def get_best_diamond_position(self, bot_pos, board):
```

  - ➢ Diamond dengan nilai tertinggi diprioritaskan.
  - ➢ Jika terdapat diamond dengan skor yang sama, maka dipilih yang paling dekat jaraknya dari posisi bot.

- **Pergerakan Menuju Tujuan: move_towards_goal**

  Setelah target ditentukan, bot akan bergerak satu langkah menuju posisi tersebut:

```
def move_towards_goal(self, bot_pos, board, board_bot):
```

  - ➢ Arah pergerakan dihitung berdasarkan selisih koordinat.
  - ➢ Validitas langkah dicek dengan fungsi is_valid_move untuk memastikan tidak terjadi tabrakan atau keluar dari batas papan.

- **Validasi Langkah: is_valid_move**

Langkah yang akan diambil diperiksa terlebih dahulu untuk memastikan keabsahannya:

```
def is_valid_move(self, x, y, board, allow_base=False, bot_base=None):
```

- ➢ Bot hanya diperbolehkan melangkah jika masih dalam area papan dan tidak menabrak objek yang tidak dapat dilewati.
- ➢ Langkah menuju base sendiri tetap diperbolehkan, meskipun terdapat objek lain di dalamnya.

- **Gerakan Alternatif: random_valid_move**

Jika tidak ditemukan langkah menuju target yang valid, maka bot akan mencoba bergerak secara acak ke arah yang diperbolehkan:

```
def random_valid_move(self, pos, board):
```

- ➢ Bot memilih arah secara acak dari empat arah utama.
- ➢ Langkah pertama yang valid akan langsung diambil.

### 2.1.2 Menjalankan Bot Program

Setelah proses implementasi logika VelocityBot selesai dilakukan, tahap selanjutnya adalah menjalankan bot dalam lingkungan permainan untuk melihat bagaimana ia beroperasi secara nyata berdasarkan strategi yang telah dirancang. Proses ini mencakup beberapa langkah penting, mulai dari penempatan file hingga pengamatan perilaku bot saat permainan berlangsung.

Pertama-tama, file program yang memuat kelas VelocityBot harus diletakkan pada direktori yang sesuai, yakni di dalam folder game/logic/. File tersebut diberi nama velocitybot.py dan berisi deklarasi kelas VelocityBot yang merupakan turunan dari kelas BaseLogic. Penempatan file yang benar akan memungkinkan sistem permainan untuk mengenali dan menjalankan bot tersebut secara otomatis ketika permainan dimulai.

Langkah berikutnya adalah menjalankan permainan melalui berkas utama proyek, yaitu main.py. Berkas ini berfungsi untuk memuat seluruh elemen dalam permainan, termasuk papan permainan (board), objek diamond, posisi base, dan tentu saja bot yang telah dikembangkan. Untuk menjalankan permainan, digunakan perintah sederhana melalui terminal.

Setelah perintah dijalankan, simulasi permainan akan dimulai dan VelocityBot akan langsung mengambil keputusan secara otomatis pada setiap giliran. Bot akan membaca posisi

dirinya saat ini, kemudian mencari diamond yang paling optimal untuk diambil, berdasarkan nilai tertinggi dan jarak terdekat—sesuai prinsip strategi greedy. Jika kapasitas penyimpanan diamond pada bot telah mencapai batas maksimum, bot akan mengubah arah menuju base untuk menyetorkan diamond yang telah dikumpulkan. Setelah proses penyetoran selesai, bot kembali aktif mencari diamond lainnya.

Selama proses permainan berlangsung, bot akan memberikan informasi secara real-time melalui output terminal. Informasi tersebut mencakup posisi bot saat ini, jumlah diamond yang dibawa, status perjalanan ke base, dan posisi target diamond. Hal ini sangat berguna dalam proses pengamatan dan evaluasi, karena pengembang dapat melihat apakah bot telah berperilaku sesuai dengan logika yang diinginkan.

# BAB III

# APLIKASI STRATEGI GREEDY

## 3.1 Proses mapping

Dalam permainan pengumpulan diamond, mapping merujuk pada cara bot membaca, memahami, dan menavigasi lingkungan sekitarnya berdasarkan posisi dirinya, diamond, dan base.

1. **Pemetaan Awal (Initial Mapping)**
   Bot akan mengakses berbagai informasi dari parameter board dan board_bot seperti:
   - Lokasi bot saat ini (board_bot.position)
   - Letak base atau markas (board_bot.properties.base)
   - Jumlah diamond yang sedang dibawa (board_bot.properties.diamonds)
   - Kapasitas maksimum yang dapat dibawa (board_bot.properties.inventory_size)
   - Semua objek di papan permainan melalui (board.game_objects)

   Semua fungsi ini digunakan dalam fungsi next_move() untuk menganalisis situasi sebelum bergerak.

2. **Menentukan Arah Tujuan**

   Bot memiliki dua kondisi dalam menentukan arah:

   - Jika muatan penuh, maka bot diarahkan untuk kembali ke base
   - Jika masih bisa menampung diamond, maka bot akan mencari diamond terbaik

   Penentuan tujuan dilakukan dengan mengatur variabel self.goal_position, baik itu mengarah ke base maupun ke diamond.

3. **Pemilihan Diamond Terbaik (Greedy Mapping)**

   Proses ini dijalankan di dalam fungsi get_best_diamond_position(). Bot:

   - Memindai semua diamond yang terlihat di papan
   - Menghitung nilai (*score*) dan jarak terdekat dari posisinya saat ini
   - Memilih diamond dengan nilai tertinggi, dan bila ada yang setara, maka dipilih yang jaraknya paling dekat

   Ini adalah pendekatan greedy, karena bot selalu memilih target dengan hasil maksimal dan waktu tempuh minimum.

4. **Perencanaan Langkah (Path Mapping)**

   Setelah tujuan ditetapkan, bot mulai merencanakan langkah menuju ke sana menggunakan fungsi move_towards_goal():

   - Menghitung pergerakan arah berdasarkan perbedaan posisi
   - Mengecek validitas langkah melalui is_valid_move(), agar tidak keluar batas/menabrak objek lain
   - Jika langkah langsung tidak bisa dilakukan, maka bot mencoba alternatif secara acak dengan random_valid_move()

   Mapping di sini bertujuan memastikan setiap langkah yang diambil bot adalah sah dan tidak membahayakan.

5. **Langkah Acak Bila Tidak Ada Tujuan**

   Jika tidak ada diamond yang terdeteksi atau tidak ada langkah aman menuju tujuan, maka bot akan mengambil langkah secara acak yang masih valid:

- Mengevaluasi semua arah (atas, bawah, kanan, kiri)
- Mengacak urutan arah dan memilih satu yang diperbolehkan

  Ini merupakan mekanisme cadangan agar bot tidak diam di tempat.

### 3.2 Eksplorasi Alternatif Solusi Greedy

Mengoptimalkan pengumpulan diamond dengan cara yang efisien dan adaptif terhadap kondisi sekitar, termasuk rintangan maupun keberadaan bot lain.

1. **Greedy dengan Rasio Nilai terhadap Jarak (Strategi X)**
   Memilih diamond berdasarkan perbandingan antara nilai score dan jarak tempuh, bukan berdasarkan salah satu saja.

   ```
   nilai_greedy = skor / jarak
   ```

   Keunggulan:

- Mempertimbangkan efisiensi langkah.
- Lebih cenderung mengambil diamond yang mudah dijangkau dan bernilai tinggi.

Kekurangan:

- Bisa mengabaikan diamond bernilai besar yang sedikit lebih jauh.
- Risiko terlalu fokus di area sempit jika banyak diamond kecil.

Ilustrasi:

Diamond A: score = 3, jarak = 2 → rasio = 1.5
Diamond B: score = 5, jarak = 5 → rasio = 1.0
→ Maka, Diamond A lebih diprioritaskan.

2. **Greedy Berbasis Area Prioritas (Strategi Y)**

   Bot menganalisis wilayah-wilayah tertentu untuk mencari area dengan konsentrasi diamond tertinggi, kemudian bergerak menuju pusatnya.

   Mekanisme:

- Peta dibagi ke dalam blok-blok kecil (contohnya 3x3).
- Hitung jumlah diamond dalam tiap blok.
- Tentukan blok dengan isi terbanyak.
- Arahkan bot ke pusat blok tersebut.

   Kelebihan:

- Mengurangi pergerakan tidak efisien.
- Memungkinkan bot berada di posisi strategis.

   Kelemahan:

- Memerlukan proses analisis lebih dalam.
- Kurang efektif jika distribusi diamond merata.

   Maka bot akan menuju ke area dengan konsentrasi diamond tertinggi.

3. **Greedy yang Menyesuaikan Pergerakan Lawan (Strategi Z)**

   Menghindari kompetisi langsung dengan bot lain dengan cara memperkirakan pergerakan mereka dan memilih target alternatif.

   Cara Kerja:

- Periksa posisi bot lain (jika tersedia).
- Estimasi siapa yang lebih dekat ke setiap diamond.

- Fokus ke diamond yang tidak dalam jangkauan cepat bot lawan.

  Kelebihan:

- Mengurangi kemungkinan gagal mendapat diamond.
- Memungkinkan pengambilan target yang lebih "aman".

  Kekurangan:

- Bergantung pada data musuh.
- Kurang efektif jika lawan tidak konsisten arah geraknya.

### 3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setiap pendekatan greedy yang telah dikaji sebelumnya memiliki karakteristik performa yang berbeda-beda, tergantung pada kondisi dan situasi permainan. Di bawah ini merupakan analisa terkait kekuatan dan kelemahan masing-masing metode:

1. **Metode X – Menghitung Rasio Nilai terhadap Jarak**

   Strategi ini mengarahkan bot untuk mengejar diamond dengan rasio terbaik antara nilai (score) dan jarak tempuh. Tujuannya adalah memaksimalkan keuntungan dengan usaha seminimal mungkin.

   **Kinerja**:

- Perhitungan sangat sederhana sehingga prosesnya cepat dan efisien secara komputasi.
- Ideal digunakan saat diamond tersebar luas dan tidak terkonsentrasi di satu area.

  **Efektivitas**:

- Mampu menghasilkan perolehan score yang konsisten dengan jumlah langkah minimal.
- Namun, kurang responsif terhadap kondisi yang berubah-ubah, seperti persaingan dengan bot lain atau pergerakan objek di peta.

2. **Metode Y – Berbasis Kepadatan Lokasi**

   Pendekatan ini mengutamakan wilayah yang memiliki banyak diamond dalam radius tertentu. Bot akan bergerak menuju area tersebut untuk mengoptimalkan hasil penambangan tanpa harus sering berpindah tempat.

**Kinerja**:

- Perhitungan lebih kompleks, karena memerlukan deteksi pola sebaran diamond dan pemilahan area padat.
- Tidak cocok jika kecepatan perhitungan menjadi prioritas utama.

**Efektivitas**:

- Sangat efisien bila diamond terkumpul dalam klaster atau kelompok.
- Memberikan peluang pengumpulan berturut-turut dalam satu area.
- Kurang optimal bila diamond tersebar merata di seluruh papan.

3. **Metode Z – Menghindari Persaingan Langsung**

   Strategi ini mempertimbangkan lokasi dan potensi gerakan bot lawan. Bot akan menghindari area yang kemungkinan besar akan diambil oleh pemain lain untuk mengurangi risiko kehilangan target.

   **Kinerja**:

- Pemrosesan lebih berat, karena mempertimbangkan banyak variabel tambahan seperti posisi dan arah gerak bot lain.
- Mungkin menyebabkan pengambilan keputusan sedikit lebih lambat.

**Efektivitas**:

- Sangat efektif dalam situasi ramai atau kompetitif karena meminimalisir rebutan diamond.
- Namun strategi ini bisa menjadi terlalu hati-hati dan kehilangan kesempatan emas karena terlalu menghindari risiko.

### 3.4 Strategi Greedy yang dipilih

Dari berbagai pendekatan greedy yang telah kami eksplorasi sebelumnya, strategi yang akhirnya kami pilih adalah metode perbandingan nilai terhadap jarak (value-to-distance ratio). Strategi ini dinilai paling optimal karena mampu memberikan hasil pengambilan keputusan yang cepat sekaligus menghasilkan perolehan poin yang maksimal dalam waktu singkat.

Metode ini mengutamakan diamond dengan rasio nilai dibanding jarak yang paling tinggi. Artinya, meskipun ada diamond yang bernilai tinggi, bot tidak akan mengejarnya jika jaraknya terlalu jauh dan tidak efisien secara langkah. Sebaliknya, diamond dengan rasio terbaik akan menjadi target utama, karena mampu memberikan hasil terbaik dalam jumlah langkah seminimal mungkin.

Keunggulan lain dari pendekatan ini adalah sifatnya yang adaptif dan ringan, sehingga cocok diterapkan pada berbagai situasi permainan tanpa membebani perhitungan logika bot. Dengan mempertimbangkan efektivitas dan efisiensi secara keseluruhan, strategi rasio nilai terhadap jarak dipilih sebagai solusi greedy utama dalam pengembangan bot ini.

# BAB IV

# IMPLEMENTASI DAN PENGUJIAN

## 4.1 Implementasi Algoritma Greedy

Pada tahap ini, algoritma **greedy** yang telah dirancang sebelumnya diimplementasikan ke dalam program bot bernama VelocityBot. Algoritma greedy digunakan karena memiliki prinsip pemilihan solusi terbaik secara lokal pada setiap langkah, dengan harapan dapat menghasilkan solusi global yang optimal. Dalam konteks permainan, solusi terbaik lokal yang dimaksud adalah diamond dengan skor tertinggi dan jarak terdekat dari posisi bot saat ini. Strategi greedy ini diimplementasikan pada fungsi-fungsi utama seperti:

1. next_move(): Menentukan langkah selanjutnya,
2. get_best_diamond_position(): Memilih target diamond terbaik berdasarkan nilai dan jarak,
3. move_towards_goal(): Mengarahkan pergerakan bot ke tujuan
4. handle_return_to_base(): Mengatur kondisi saat inventori penuh,
5. random_valid_move(): Menangani kondisi jika tidak ada target valid.

### 4.1.1 Pseudocode

```
CLASS VelocityBot EXTENDS BaseLogic

  # Inisialisasi bot dengan goal_position dan status kembali ke base
  FUNCTION __init__():
    SET goal_position TO None            # Tujuan posisi bot saat ini
    SET returning_to_base TO False       # Status apakah bot sedang kembali ke base

  # Fungsi utama untuk menentukan gerakan berikutnya
  FUNCTION next_move(board_bot, board):
    SET bot_pos TO board_bot.position
    SET inventory TO jumlah diamond di board_bot (default 0)
    SET inventory_limit TO kapasitas maksimum inventory (default 5)
    SET base_pos TO posisi base bot

    # Jika bot sedang kembali ke base untuk setor diamond
    IF returning_to_base IS True:
      IF bot_pos IS base_pos:
        # Jika sudah sampai base, setor diamond dan cari tujuan diamond baru
        SET diamond di board_bot TO 0
        SET returning_to_base TO False
        SET goal_position TO get_best_diamond_position(bot_pos, board)
        # Jika tidak ada diamond lain di sekitar, lakukan gerakan acak
        IF goal_position EQUALS bot_pos:
          RETURN random_valid_move(bot_pos, board)
        ELSE:
          # Gerak menuju diamond yang dipilih
          RETURN move_towards_goal(bot_pos, board, board_bot)
      ELSE:
        # Jika belum sampai base, terus gerak ke base
        SET goal_position TO base_pos
        RETURN move_towards_goal(bot_pos, board, board_bot)
```

```
    # Jika inventory sudah penuh, mulai kembali ke base
    IF inventory >= inventory_limit:
      SET returning_to_base TO True
      SET goal_position TO base_pos
      RETURN move_towards_goal(bot_pos, board, board_bot)

    # Jika belum penuh, cari diamond terbaik sebagai tujuan
    SET goal_position TO get_best_diamond_position(bot_pos, board)

    # Jika bot sudah di posisi diamond, lakukan gerakan acak untuk eksplorasi
    IF goal_position EQUALS bot_pos:
      RETURN random_valid_move(bot_pos, board)

    # Gerak ke arah diamond yang sudah ditentukan
    RETURN move_towards_goal(bot_pos, board, board_bot)

  # Fungsi untuk menggerakkan bot menuju goal_position
  FUNCTION move_towards_goal(bot_pos, board, board_bot):
    IF goal_position IS NOT None AND goal_position != bot_pos:
      # Hitung arah gerak dari posisi saat ini ke tujuan
      CALCULATE direction (dx, dy) TO goal_position DARI bot_pos

      SET new_x TO bot_pos.x + dx
      SET new_y TO bot_pos.y + dy

      # Cek apakah gerakan valid (tidak menabrak rintangan kecuali base)
      IF is_valid_move(new_x, new_y, board, allow_base=True, bot_base=board_bot.properties.base):
        RETURN (dx, dy)  # Gerak ke arah goal

    # Jika tidak bisa gerak ke goal, lakukan gerakan acak
    RETURN random_valid_move(bot_pos, board)
```

```
  # Fungsi untuk mencari posisi diamond terbaik berdasarkan skor dan jarak
  FUNCTION get_best_diamond_position(bot_pos, board):
    SET best_target TO None
    SET best_score TO -1
    SET best_distance TO infinity

    FOR each diamond_obj IN board.diamonds:
      # Ambil skor diamond, default 1 jika tidak ada
      SET score TO diamond_obj.properties.score (default 1)
      # Hitung jarak Manhattan dari posisi bot ke diamond
      SET dist TO jarak Manhattan antara diamond_obj.position dan bot_pos

      # Pilih diamond dengan skor tertinggi dan jarak terdekat jika skor sama
      IF score > best_score OR (score == best_score AND dist < best_distance):
        SET best_target TO diamond_obj
        SET best_score TO score
        SET best_distance TO dist

    # Kembalikan posisi diamond terbaik atau None jika tidak ada
    IF best_target IS NOT None:
      RETURN best_target.position
    ELSE:
      RETURN None

  # Fungsi untuk mengecek apakah suatu posisi bisa dilewati oleh bot
  FUNCTION is_valid_move(x, y, board, allow_base=False, bot_base=None):
    # Cek apakah koordinat berada di dalam papan
    IF x atau y di luar batas papan:
      RETURN False

    FOR each obj IN board.game_objects:
      # Cek apakah ada objek di posisi tersebut
      IF obj.position == (x, y):
        # Jika objek bukan diamond, teleport, atau tombol merah, maka halangi gerakan
        IF obj.type BUKAN salah satu ["DiamondGameObject", "TeleportGameObject", "RedButtonGameObject"]:
          # Jika diizinkan melewati base sendiri, perbolehkan
```

```
          IF allow_base IS True AND bot_base IS NOT None AND posisi (x, y) EQUALS bot_base:
            RETURN True
          ELSE:
            RETURN False

      # Jika tidak ada objek penghalang, gerakan valid
      RETURN True

    # Fungsi untuk menghasilkan gerakan acak yang valid (atas, bawah, kiri, kanan)
    FUNCTION random_valid_move(pos, board):
      DEFINE possible_moves = [(1,0), (0,1), (-1,0), (0,-1)]  # gerakan kanan, bawah, kiri, atas
      RANDOMIZE urutan possible_moves  # supaya gerakan acak

      FOR each (dx, dy) IN possible_moves:
        SET new_x TO pos.x + dx
        SET new_y TO pos.y + dy
        # Cek validitas gerakan
        IF is_valid_move(new_x, new_y, board):
          RETURN (dx, dy)

      # Jika tidak ada gerakan valid, diam saja
      RETURN (0, 0)

  END CLASS
```

### 4.1.2 Penjelasan Alur Program

Program VelocityBot merupakan implementasi bot dalam permainan pengumpulan diamond yang menggunakan logika berbasis kondisi saat ini. Bot ini dirancang dengan dua fungsi utama: mengumpulkan diamond sebanyak mungkin dan kembali ke markas ketika kapasitas penyimpanan (inventori) sudah penuh. Setiap giliran, bot akan memeriksa posisinya, jumlah diamond yang telah dikumpulkan, batas kapasitas inventori, serta lokasi base. Bila bot sedang dalam proses kembali ke base, ia akan mengecek apakah telah mencapai markas. Jika ya, diamond yang dibawa dianggap telah disetor (simulasi), status kembali di reset, dan bot langsung mencari target diamond berikutnya. Jika belum sampai, bot terus bergerak menuju base.

Jika jumlah diamond dalam inventori telah mencapai batas maksimal, bot secara otomatis beralih ke mode kembali ke base dan mengatur tujuannya ke markas untuk menyimpan diamond. Namun jika inventori belum penuh, bot akan memilih diamond yang paling menguntungkan berdasarkan kombinasi nilai tertinggi dan jarak terpendek dari posisinya. Pemilihan dilakukan melalui fungsi get_best_diamond_position, yang menilai setiap diamond di papan berdasarkan skor dan kedekatannya. Jika bot sudah berada di atas posisi diamond target, maka ia akan bergerak secara acak untuk mencegah kondisi diam di tempat.

Untuk proses perpindahan, digunakan fungsi move_towards_goal yang menentukan arah berdasarkan posisi bot dan target, serta memverifikasi validitas langkah tersebut menggunakan is_valid_move. Jika langkah tidak sah, bot akan mencari langkah alternatif yang masih dalam batas papan dan tidak bertabrakan dengan objek yang menghalangi, kecuali diamond, teleport, tombol merah, atau base (jika diperlukan). Jika tidak ada langkah valid ke arah tujuan, maka bot akan memilih langkah acak melalui fungsi random_valid_move. Program ini memungkinkan bot bergerak secara efisien dan responsif sesuai situasi permainan, walau masih mengandalkan pendekatan greedy yang sederhana. Namun, terdapat kesalahan penulisan pada konstruktor __init__, yang perlu diperbaiki agar atribut seperti goal_position dan returning_to_base dapat diinisialisasi dengan benar.

## 4.2 Struktur Data yang Digunakan

Dalam implementasi bot MyBot, terdapat sejumlah struktur data yang digunakan untuk mengatur informasi serta pengambilan keputusan selama permainan berlangsung. Berikut penjelasannya:

1. Integer (Bilangan Bulat)
   Digunakan untuk menyimpan nilai numerik seperti jumlah diamond yang dibawa (inventory) dan batas maksimal penyimpanan (inventory_limit). Selain itu, bilangan bulat juga digunakan untuk menghitung jarak antara posisi bot dan objek target menggunakan

metode jarak Manhattan.

2. Boolean (Logika True/False)
   Variabel self.returning_to_base memegang nilai boolean yang menunjukkan apakah bot sedang dalam perjalanan menuju base atau tidak. Nilai ini menjadi penentu jalannya logika utama dalam memilih strategi gerakan bot.

3. Objek Position
   Struktur data ini merepresentasikan koordinat posisi pada papan permainan, biasanya dalam format (x, y). Posisi bot dan base keduanya disimpan dalam tipe Position, sehingga memudahkan proses navigasi dan perbandingan lokasi.

4. List (Daftar)
   Daftar digunakan untuk menyimpan banyak elemen seperti semua objek dalam permainan (game_objects) dan seluruh diamond di papan (diamonds). Dalam proses pergerakan acak, daftar arah gerak juga disimpan dalam list dan diacak untuk memilih langkah yang sah secara acak.

5. Tuple
   Struktur data tuple dimanfaatkan untuk menyimpan pasangan arah gerak (dx, dy). Tuple ini kemudian dikembalikan sebagai hasil dari langkah berikutnya yang akan diambil oleh bot.

6. Objek GameObject dan Properti Dinamis
   Objek ini merepresentasikan entitas dalam permainan, termasuk bot itu sendiri. Melalui properti dinamisnya, informasi seperti lokasi base, jumlah diamond yang dimiliki, dan kapasitas tas dapat diakses dan diperbarui.

7. None untuk Penanda Awal
   Atribut self.goal_position pada awalnya diset sebagai None untuk menandai bahwa belum ada tujuan yang ditetapkan. Nantinya, nilai ini diubah menjadi koordinat tujuan seperti lokasi diamond atau base tergantung kondisi permainan.

Secara keseluruhan, struktur data yang digunakan memungkinkan bot untuk mengelola status, menentukan langkah strategis, serta menavigasi papan secara optimal berdasarkan prinsip algoritma greedy. Penggunaan struktur ini mendukung proses pengambilan keputusan yang cepat dan logis dalam permainan.

## 4.3 Pengujian Program

### 4.3.1 Skenario Pengujian

Untuk memastikan bahwa bot berjalan sesuai dengan logika yang telah dirancang, dilakukan uji coba melalui beberapa kondisi permainan yang berbeda. Uji coba ini dirancang untuk mengevaluasi bagaimana bot merespons situasi tertentu, seperti:

1. Skenario 1: Bot diharapkan dapat menemukan dan menuju diamond dengan nilai paling tinggi yang berada dalam jangkauan.

2. Skenario 2: Saat kapasitas penyimpanan sudah maksimal, bot harus beralih ke mode kembali ke base.

3. Skenario 3: Setelah berhasil mencapai base, bot harus menyimpan diamond dan kembali menjalankan misi pencarian.

4. Skenario 4: Jika tidak ada diamond yang tersedia di sekitar, bot tetap bergerak secara acak namun tetap berada dalam batas gerakan yang diperbolehkan.

### 4.3.2 Hasil Pengujian dan Analisis

Hasil Pengujian :

Setelah menguji bot dengan skenario di atas, diperoleh beberapa hasil sebagai berikut:

- Bot berhasil mendeteksi diamond dengan nilai tertinggi dan secara efisien menuju lokasinya.

- Ketika inventori mencapai batas maksimum, bot otomatis memprioritaskan perjalanan pulang ke base.

- Sesampainya di base, proses penyimpanan diamond dilakukan dengan lancar, lalu bot kembali mencari diamond lainnya.

- Jika tidak terdapat diamond yang bisa dijangkau, bot tetap aktif dengan memilih gerakan acak yang sah untuk menjaga mobilitas.

Analisis Pengujian :

Dari seluruh rangkaian uji coba, dapat disimpulkan bahwa algoritma greedy yang diterapkan cukup efektif untuk menangani pengambilan keputusan dalam permainan. Bot mampu beradaptasi dengan perubahan situasi di lapangan, seperti memutuskan kapan harus mengumpulkan dan kapan harus menyetor diamond. Selain itu, sistem juga mampu menghindari area yang tidak bisa dilalui.

Meski begitu, dalam situasi kompleks seperti jalan yang tertutup atau diamond yang sulit diakses, pendekatan greedy ini mungkin perlu dikombinasikan dengan algoritma pencarian jalur seperti BFS atau A* untuk meningkatkan efisiensi navigasi. Secara keseluruhan, strategi yang digunakan telah menunjukkan performa yang konsisten dan responsif terhadap kondisi permainan.

# BAB V

# KESIMPULAN DAN SARAN

## 5.1 Kesimpulan

Bot ini dikembangkan dengan menerapkan pendekatan greedy untuk mengoptimalkan perolehan poin dalam permainan secara cepat dan efisien. Strategi yang digunakan berfokus pada pemilihan diamond berdasarkan rasio antara nilai dan jarak (value-to-distance ratio). Dengan strategi ini, bot akan mengejar diamond yang menawarkan keuntungan maksimal dengan usaha seminimal mungkin dalam hal jumlah langkah. Ini selaras dengan prinsip greedy: membuat keputusan terbaik pada kondisi saat ini untuk memperoleh hasil mendekati optimal secara keseluruhan.

A. Dalam penerapannya, bot beroperasi dalam dua fase :

1. Fase pengumpulan diamond
   Saat inventori masih tersedia, bot akan mencari diamond yang paling menguntungkan berdasarkan rasio nilai terhadap jaraknya. Diamond yang memiliki nilai tinggi namun terlalu jauh akan diabaikan jika tidak efisien untuk dikejar.

2. Fase kembali ke markas
   Jika kapasitas inventory telah penuh, bot akan segera kembali ke base untuk menyimpan diamond, lalu melanjutkan kembali proses pencarian. Proses ini menjaga alur kerja bot tetap efisien dan terus berkontribusi terhadap peningkatan poin.

Jika tidak ada target yang dapat dijangkau atau pergerakan menuju target terhambat, bot akan melakukan langkah acak yang tetap valid untuk menghindari kondisi diam (idle) dan menjaga fleksibilitas dalam bergerak.

B. Algoritma greedy yang digunakan dalam sistem ini mengutamakan:

1. Pengambilan keputusan berbasis kondisi saat ini, tanpa memperhitungkan skenario masa depan yang lebih kompleks.

2. Pemilihan pilihan terbaik secara lokal, yaitu diamond dengan rasio nilai terhadap jarak tertinggi, untuk memperoleh hasil yang efisien dalam waktu terbatas.

3. Ringan dalam perhitungan dan fleksibel, sehingga cocok digunakan dalam permainan real-time yang menuntut respons cepat.

4. Tidak bergantung pada perhitungan rute kompleks seperti pathfinding, menjadikan implementasinya lebih sederhana namun tetap efektif.

Dengan pendekatan ini, bot mampu melakukan pengambilan keputusan secara cepat, hemat sumber daya, dan tetap adaptif terhadap perubahan situasi dalam permainan. Strategi ini sangat tepat untuk bot dalam game berbasis pengumpulan sumber daya secara kompetitif.

## 5.2 Saran

Bot VelocityBot masih memiliki ruang untuk ditingkatkan agar kinerjanya lebih optimal. Salah satu hal yang perlu diperhatikan adalah perbaikan pada bagian inisialisasi, di mana fungsi konstruktor masih dituliskan secara tidak tepat sehingga dapat menyebabkan atribut penting tidak terbentuk dengan benar. Selain itu, mekanisme pemilihan diamond dapat ditingkatkan dengan menggunakan pendekatan rasio antara nilai dan jarak, guna memastikan target yang dipilih benar-benar efisien untuk dikejar.

Selanjutnya, strategi kembali ke base sebaiknya dibuat lebih adaptif, tidak hanya bergantung pada kapasitas inventory yang penuh, tetapi juga mempertimbangkan faktor seperti jarak ke base dan potensi diamond di sekitarnya. Mekanisme pergerakan acak yang digunakan saat tidak ada target juga dapat dikembangkan menjadi strategi eksplorasi yang lebih terarah. Selain itu, kemampuan bot dalam memperhitungkan keberadaan pemain lain maupun objek pengganggu di dalam arena juga perlu ditambahkan untuk meningkatkan kecerdasan pengambilan keputusan. Terakhir, penggunaan sistem pencatatan (logging) yang lebih terstruktur disarankan untuk menggantikan perintah print, agar proses pemantauan dan evaluasi performa bot dapat dilakukan dengan lebih profesional.

# LAMPIRAN

## A. Repository Github (link)

Link : Tubes1_VELOCITY

## B. Video Penjelasan (Link Gdrive)

Link: Link Gdrive (VELOCITY)

## DAFTAR PUSTAKA

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Cambridge, MA:MIT Press. (2009). Introduction to Algorithms. In (3rd ed ed.).

[2] S. Dasgupta, & U. Vazirani. (2008). Algorithms. In *Algorithms* (1st ed ed.). New York: McGraw-Hill.