IMPLEMENTASI ALGORITMA X 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 TrioBRE

Rifael Eurico Sitorus	123140077
Ebentua Philippus Limbong	123140086
Muhammad Bintang Al-Fasya	123140098

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Diamonds merupakan suatu programming challenge yang mempertandingkan kemampuan bot yang telah dibuat dengan bot dari para pemain lainnya. Tujuan utama dalam permainan Diamonds adalah mengumpulkan diamond sebanyak mungkin dalam waktu yang telah ditentukan. Setiap bot memiliki inventory dengan kapasitas terbatas dan harus kembali ke base untuk menyimpan diamond yang telah dikumpulkan agar score bertambah.

Permainan Diamonds ini terdiri dari beberapa komponen utama:

1. **Diamonds**: Terdapat dua jenis diamond yaitu diamond biru (1 poin) dan diamond merah (2 poin). Diamond akan di-regenerate secara otomatis jika seluruh diamond pada board sudah habis.
2. **Red Button**: ketika bot melangkahi red button, maka semua diamond akan di-generate kembali pada board dengan posisi acak. Posisi red button juga akan berubah secara acak.
3. **Teleporters**: Terdapat 2 teleporter yang saling terhubung. Bot yang melewati teleporter akan berpindah ke posisi teleporter lainnya.
4. **Bots and Bases**: Setiap bot memiliki base untuk menyimpan diamond. Ketika bot mencapai base, score bertambah sesuai nilai diamond yang dibawa dan inventory menjadi kosong.
5. **Inventory**: Bot memiliki inventory dengan kapasitas maksimum 5 diamond sebagai tempat penyimpanan sementara.
6. **Tackle System**: Bot dapat melakukan tackle terhadap bot lain. Bot yang di-tackle akan kehilangan semua diamond di inventory dan dikirim kembali ke base.

Aturan-aturan yang tedapat dari permainan ini yaitu :

- Setiap bot ditempatkan secara random di board dengan score dan inventory awal bernilai nol
- Bot bergerak secara bergantian dengan waktu yang sama untuk setiap pemain
- Bot harus menghindari bertemu dengan bot lawan untuk mencegah tackle
- Permainan berakhir ketika waktu habis dan score tertinggi menjadi pemenang

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy merupakan salah satu paradigma dalam pemecahan masalah yang bekerja dengan cara membuat keputusan optimal secara lokal pada setiap langkah dengan harapan keputusan tersebut akan mengarah pada solusi global yang optimal. Pendekatan ini melibatkan proses pemilihan elemen yang dianggap paling menguntungkan pada setiap iterasi tanpa meninjau konsekuensi dari pilihan tersebut pada langkah selanjutnya.

Secara umum, algoritma greedy terdiri dari beberapa langkah utama, yaitu:

- Inisialisasi, yaitu menentukan kondisi awal dan struktur data yang diperlukan.
- Seleksi, memilih elemen terbaik yang tersedia pada saat itu berdasarkan suatu kriteria tertentu (misalnya nilai tertinggi atau biaya terendah).
- Feasibility Check, memastikan bahwa elemen yang dipilih tidak melanggar batasan atau kendala dalam masalah.
- Konstruksi Solusi, menambahkan elemen yang valid ke dalam solusi parsial.
- Iterasi, mengulangi proses hingga solusi akhir terbentuk.

Pendekatan greedy sangat efektif untuk masalah yang memiliki dua properti penting, yaitu greedy choice property dan optimal substructure. Greedy choice property menyatakan bahwa pilihan lokal yang optimal akan menghasilkan solusi global yang optimal, sementara optimal substructure menunjukkan bahwa solusi optimal dari suatu masalah mengandung solusi optimal dari submasalah-submasalahnya.

Beberapa contoh permasalahan yang dapat diselesaikan secara optimal menggunakan algoritma greedy antara lain adalah masalah pencarian pohon rentang minimum (Minimum Spanning Tree) dengan algoritma Kruskal dan Prim, pengkodean Huffman (Huffman Coding), pemilihan aktivitas (Activity Selection Problem), dan masalah ransel pecahan (Fractional Knapsack Problem).

Meskipun algoritma greedy tergolong efisien dalam hal waktu dan memori, pendekatan ini tidak selalu menjamin solusi optimal untuk semua jenis masalah. Oleh karena itu, sebelum menerapkan algoritma greedy, penting untuk memastikan bahwa permasalahan yang dihadapi memenuhi kriteria yang sesuai.

2.2 Cara Kerja Program

Program ini bekerja sebagai sistem permainan berbasis bot di mana setiap bot dikendalikan oleh logika tertentu yang dapat dikustomisasi. Dalam konteks ini, algoritma greedy diimplementasikan ke dalam logika bot untuk menentukan langkah terbaik berdasarkan kondisi terkini di papan permainan (game board). Bot akan bergerak dalam grid dua dimensi, dengan tujuan utama seperti mengumpulkan diamond sebanyak mungkin atau menyelesaikan objektif tertentu dengan efisien.

Program terdiri atas dua komponen utama: game engine yang mensimulasikan dunia permainan dan bot sebagai agen cerdas yang berinteraksi dalam lingkungan tersebut.

1. **Cara Implementasi Program**

   Cara kerja program ini adalah dengan memanfaatkan arsitektur client-server, di mana game engine bertindak sebagai server yang mengatur seluruh jalannya permainan, dan bot sebagai client yang melakukan aksi berdasarkan strategi yang telah diprogram.

   Bot dikembangkan dengan cara membuat kelas baru dalam direktori /game/logic/ yang mewarisi (inherit) kelas BaseLogic. Di dalam kelas ini, programmer mengimplementasikan method next_move, yaitu method yang akan dipanggil secara berkala oleh sistem untuk menentukan langkah selanjutnya. Nilai yang dikembalikan dari next_move berupa arah gerakan (delta_x, delta_y), seperti (0,1) untuk ke atas atau (1,0) untuk ke kanan. Dalam implementasi greedy, logika dalam method ini akan memilih langkah yang memberikan keuntungan maksimal secara langsung pada saat itu, seperti mendekati diamond terdekat.

2. Menjalankan Game Engine

Sebelum menjalankan bot, game engine harus terlebih dahulu dikonfigurasi. Berikut langkah-langkahnya:

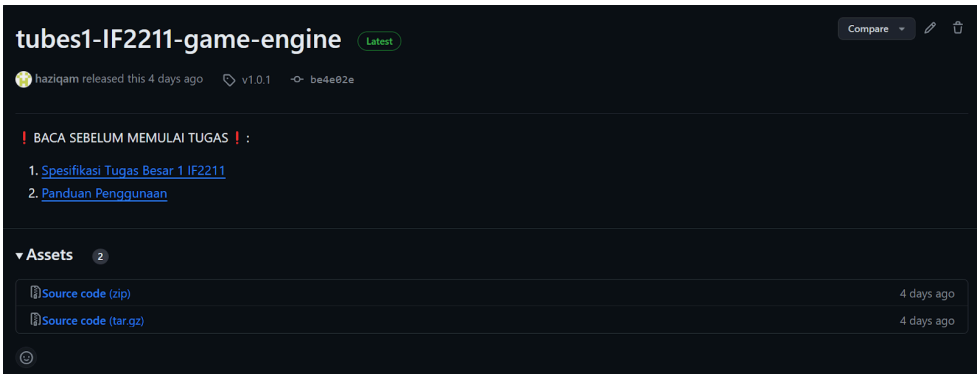
a. *Requirement* yang harus di-install

- Node.js (https://nodejs.org/en)
- Docker desktop (https://www.docker.com/products/docker-desktop/)
- Yarn

```
npm install --global yarn
```

b. Instalasi dan konfigurasi awal

1) Download source code (.zip) pada release game engine



2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal

3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-game-engine-1.1.0
```

4) Install dependencies menggunakan Yarn

```
yarn
```

5) Setup default environment variable dengan menjalankan script berikut

Untuk Windows

```
./scripts/copy-env.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/copy-env.sh
./scripts/copy-env.sh
```

6) Setup local database (buka aplikasi docker desktop terlebih dahulu, lalu jalankan command berikut di terminal)

```
docker compose up -d database
```

Lalu jalankan script berikut. Untuk Windows

```
./scripts/setup-db-prisma.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/setup-db-prisma.sh
./scripts/setup-db-prisma.sh
```

Build dan run game engine dengan:

a.

```
npm run build
```

b. *Run*

```
npm run start
```

Game dapat diakses melalui http://localhost:8082/.

3. Menjalankan Bot Program

Untuk menjalankan bot, langkah-langkah yang dilakukan adalah sebagai berikut:

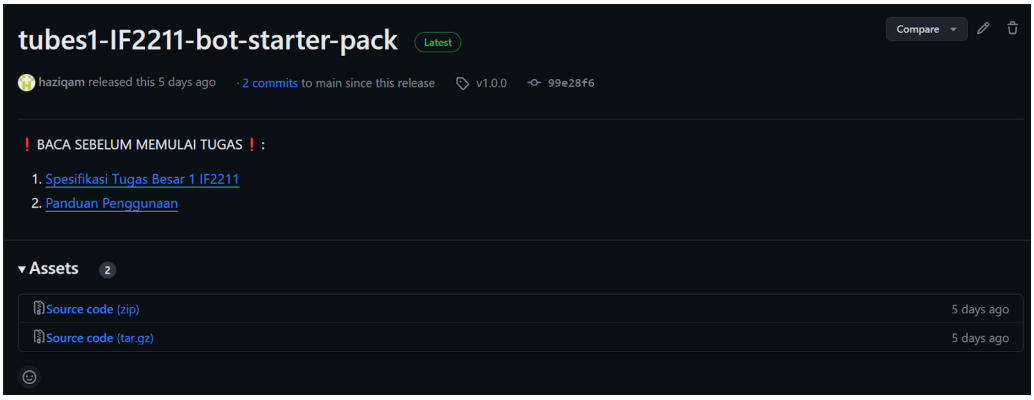
Setiap bot yang dijalankan akan muncul di tampilan frontend dan beraksi selama durasi permainan berlangsung. Setelah permainan selesai, skor akhir akan ditampilkan.

a. *Requirement* yang harus di-install

- Python (https://www.python.org/downloads/)

b. Instalasi dan konfigurasi awal

1) Download source code (.zip) pada release bot starter pack



2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal

3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-bot-starter-pack-1.0.1
```

4) Install dependencies menggunakan pip

```
pip install -r requirements.txt
```

c. *Run*

Untuk menjalankan satu bot (pada contoh ini, kita menjalankan satu bot dengan logic yang terdapat pada file game/logic/random.py)

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

Untuk menjalankan beberapa bot sekaligus (pada contoh ini, kita menjalankan 4 bot dengan logic yang sama, yaitu game/logic/random.py)

- Untuk windows

```
./run-bots.bat
```

- Untuk Linux / (possibly) macOS

```
./run-bots.sh
```

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses Mapping

Proses mapping dalam permainan Diamonds melibatkan pemetaan kondisi board game ke dalam struktur data yang dapat diproses oleh algoritma greedy. Dalam implementasi DiamondMasterRebalanced, mapping dilakukan terhadap beberapa elemen penting:

1. **Pemetaan Posisi dan Objek**: Setiap objek di board (diamond, bot, teleporter, red button, base) dipetakan ke dalam koordinat (x, y) pada grid. Posisi ini direpresentasikan menggunakan class Position yang memiliki atribut x dan y.

2. **Pemetaan Nilai Diamond**: Diamond merah dipetakan dengan nilai 2, diamond biru dengan nilai 1. Fungsi _is_red_diamond() digunakan untuk mengidentifikasi jenis diamond.
3. **Pemetaan Area:** Board dibagi menjadi dua area utama:

    - Home Area: Area di sekitar base dengan radius tertentu (home_area_radius = 8). Diamond dalam area ini diprioritaskan.
    - Far Area: Area di luar home area. Diamond di area ini hanya diambil jika home area kosong.

4. **Pemetaan Cluster Diamond**: Diamond dikelompokkan dalam cluster berdasarkan kedekatan posisi menggunakan region grid 4x4. Fungsi _update_diamond_clusters() memetakan diamond ke dalam cluster untuk analisis kepadatan.

5. **Pemetaan Teleporter**: Teleporter dipetakan sebagai pasangan yang saling terhubung. Fungsi _identify_teleporter_pairs() mengidentifikasi pasangan teleporter untuk navigasi yang lebih efisien.

6. Pemetaan Prioritas: Setiap target potensial (diamond, base, teleporter) dipetakan dengan nilai prioritas berdasarkan beberapa faktor:

    - Jarak dari posisi bot saat ini (Manhattan distance)
    - Jarak dari base
    - Nilai diamond (merah = 2, biru = 1)

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam pengembangan bot untuk permainan Diamonds, kami mengeksplorasi beberapa alternatif strategi greedy:

1. **Greedy Berdasarkan Jarak ke Diamond (Nearest Diamond First):**

    Deskripsi: Bot selalu memilih diamond terdekat dari posisinya saat ini.
    Kriteria Greedy: Jarak Manhattan minimum ke diamond.
    Kelebihan: Sederhana dan efisien untuk mengumpulkan diamond dalam jarak dekat.
    Kekurangan: Tidak mempertimbangkan jarak kembali ke base, sehingga bisa tidak efisien untuk perjalanan pulang-pergi.

2. **Greedy Berdasarkan Jarak ke Home (Home-Centric):**

    Deskripsi: Bot selalu memilih diamond terdekat dari home base.
    Kriteria Greedy: Jarak Manhattan minimum dari diamond ke home base.
    Kelebihan: Memastikan efisiensi perjalanan pulang ke base.
    Kekurangan: Mungkin mengabaikan diamond yang dekat dengan posisi bot saat ini.

3. **Greedy Berdasarkan Nilai Diamond (Value-Based):**

Deskripsi: Bot memprioritaskan diamond merah (2 poin) dibandingkan diamond biru (1 poin).
Kriteria Greedy: Nilai diamond dibagi dengan jarak.
Kelebihan: Memaksimalkan nilai poin yang dikumpulkan.
Kekurangan: Mungkin tidak efisien dalam hal jarak tempuh.

4. **Greedy Hybrid (Kombinasi Jarak dan Nilai):**

Deskripsi: Bot mempertimbangkan kombinasi jarak dan nilai diamond.
Kriteria Greedy: (Nilai diamond) / (Jarak ke diamond + Jarak dari diamond ke base).
Kelebihan: Menyeimbangkan nilai dan efisiensi perjalanan.
Kekurangan: Lebih kompleks untuk diimplementasikan.

5. **Greedy dengan Mode Switching (Implementasi Saat Ini):**

Deskripsi: Bot beralih antara mencari diamond terdekat dari home dan mencari diamond terdekat dari posisi saat ini.
Kriteria Greedy: Jarak Manhattan minimum, dengan switching mode berdasarkan kondisi.
Kelebihan: Menggabungkan efisiensi dari dua pendekatan berbeda.
Kekurangan: Perlu penanganan khusus untuk switching mode.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Untuk menganalisis efisiensi dan efektivitas solusi greedy yang telah dieksplor, kami melakukan perbandingan berdasarkan beberapa kriteria:

1. *Nearest Diamond First*:

- Efisiensi: O(n) untuk setiap langkah, di mana n adalah jumlah diamond di board.
- Efektivitas: Baik untuk pengumpulan diamond jarak pendek, tetapi tidak optimal untuk perjalanan pulang-pergi.
- Skenario Terbaik: Board dengan diamond tersebar merata.

- Skenario Terburuk: Diamond terkonsentrasi jauh dari base, menyebabkan perjalanan pulang yang tidak efisien.

2. *Home-Centric*:

- Efisiensi: O(n) untuk setiap langkah.
- Efektivitas: Optimal untuk meminimalkan jarak total perjalanan pulang-pergi.
- Skenario Terbaik: Diamond terkonsentrasi dekat dengan base.
- Skenario Terburuk: Diamond tersebar jauh dari base, menyebabkan perjalanan pengumpulan yang tidak efisien.

3. *Value-Based*:

- Efisiensi: O(n) untuk setiap langkah.
- Efektivitas: Maksimal dalam hal nilai poin yang dikumpulkan, tetapi mungkin tidak efisien dalam hal jarak.
- Skenario Terbaik: Diamond merah tersebar merata dan mudah diakses.
- Skenario Terburuk: Diamond merah terkonsentrasi jauh dari base dan posisi bot.

4. Greedy Hybrid:

- Efisiensi: O(n) untuk setiap langkah, dengan perhitungan yang sedikit lebih kompleks.
- Efektivitas: Menyeimbangkan nilai dan efisiensi perjalanan.
- Skenario Terbaik: Campuran diamond merah dan biru dengan distribusi yang bervariasi.
- Skenario Terburuk: Kondisi ekstrem di mana trade-off antara nilai dan jarak sulit ditentukan.

5. Mode Switching (Implementasi Saat Ini):

- Efisiensi: O(n) untuk setiap langkah, dengan overhead kecil untuk switching mode.
- Efektivitas: Menggabungkan kelebihan dari pendekatan *Nearest Diamond First* dan *Home-Centric*.

- Skenario Terbaik: Diamond tersebar dalam cluster-cluster, di mana bot dapat mengumpulkan beberapa diamond dalam satu perjalanan.
- Skenario Terburuk: Distribusi diamond yang sangat dinamis dan berubah cepat, menyebabkan switching mode yang terlalu sering.

Berdasarkan analisis di atas, strategi Mode *Switching* menawarkan keseimbangan terbaik antara efisiensi dan efektivitas untuk berbagai skenario permainan Diamonds.

3.4 Strategi Greedy yang Dipilih

Setelah menganalisis berbagai alternatif, strategi greedy yang dipilih untuk implementasi *BotGreedy* adalah *Greedy* dengan Mode *Switching*. Strategi ini dipilih karena:

1. **Adaptabilitas**: Strategi ini dapat beradaptasi dengan berbagai kondisi board, baik ketika diamond terkonsentrasi dekat base maupun tersebar jauh.
2. **Efisiensi Cluster Collection**: Dengan beralih ke mode pencarian berdasarkan posisi saat ini setelah menemukan diamond pertama, bot dapat mengumpulkan cluster diamond secara efisien sebelum kembali ke base.
3. **Optimasi Perjalanan**: Bot hanya kembali ke base ketika inventory penuh, memaksimalkan efisiensi pengumpulan.
4. **Penanganan Kasus Khusus**: Strategi ini memiliki penanganan khusus untuk kondisi seperti inventory hampir penuh (menghindari diamond merah jika hanya tersisa 1 slot) dan kondisi macet.
5. **Keseimbangan Kompleksitas dan Performa**: Meskipun sedikit lebih kompleks dari strategi sederhana, implementasinya tetap efisien dengan kompleksitas O(n) untuk setiap langkah.

Implementasi strategi ini melibatkan dua mode utama:

- Mode *Home-Based* (mode_kumpul = *False*): Mencari diamond terdekat dari home base, optimal untuk memulai pengumpulan.

- Mode *Position-Based* (mode_kumpul = *True*): Mencari diamond terdekat dari posisi saat ini, optimal untuk mengumpulkan cluster diamond.

Switching antara dua mode ini dilakukan berdasarkan kondisi:

- Setelah menemukan diamond terdekat dari home, beralih ke mode position-based.
- Jika tidak ada diamond di sekitar posisi saat ini, kembali ke mode home-based.
- Setelah kembali ke base, reset ke mode home-based.

Strategi ini juga dilengkapi dengan mekanisme anti-stuck untuk mengatasi kondisi terjebak dan logika untuk menggunakan red button ketika diamond habis.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Metode Utama

Metode next_move adalah metode utama yang dipanggil pada setiap langkah permainan. Metode ini mengimplementasikan logika pengambilan keputusan utama:

```
def next_move(self, bot: GameObject, papan: Board) -> Tuple[int,
int]:
    posisi_saat_ini = bot.position
    properti = bot.properties

    # Deteksi jika bot terjebak
    if self.posisi_terakhir and position_equals(posisi_saat_ini,
self.posisi_terakhir):
```

```
        self.hitung_macet += 1
    else:
        self.hitung_macet = 0
    self.posisi_terakhir = posisi_saat_ini

    # Jika macet lebih dari 3 langkah, coba keluar
    if self.hitung_macet >= 3:
        return self._keluar_dari_macet(posisi_saat_ini, papan)

    # Kembali ke home hanya jika tas sudah penuh
    if properti.diamonds >= properti.inventory_size:
        self.mode_kumpul = False  # Reset ke mode pencarian dari
home setelah kembali
        return self._bergerak_menuju(posisi_saat_ini, properti.base,
papan)

    # Tentukan target diamond tergantung mode
    if self.mode_kumpul:
        # Cari diamond terdekat dari posisi sekarang
        target_diamond =
self._cari_diamond_terdekat_dari_posisi(papan, properti,
posisi_saat_ini)
        if not target_diamond:
            # Tidak ada di sekitar, kembali ke pencarian dari home
            self.mode_kumpul = False
            target_diamond =
self._cari_diamond_terdekat_dari_home(papan, properti)
    else:
        # Cari diamond terdekat dari home
        target_diamond =
self._cari_diamond_terdekat_dari_home(papan, properti)
        if target_diamond:
            self.mode_kumpul = True  # Setelah menemukan, pindah ke
mode pencarian dari posisi

    if target_diamond:
```

```
        return self._bergerak_menuju(posisi_saat_ini,
target_diamond.position, papan)

    # Gunakan tombol merah jika diamond sudah habis
    if len(papan.diamonds) == 0:
        self.mode_kumpul = False
        tombol_merah = self._cari_tombol_merah(papan)
        if tombol_merah:
            return self._bergerak_menuju(posisi_saat_ini,
tombol_merah.position, papan)

    # Kembali ke home jika masih membawa diamond meski sudah tidak
ada sisa
    if properti.diamonds > 0:
        self.mode_kumpul = False
        return self._bergerak_menuju(posisi_saat_ini, properti.base,
papan)

    return (0, 0)
```

2. **Metode Pencarian Diamond terdekat dari Home**

```
def _cari_diamond_terdekat_dari_home(self, papan: Board, properti)
-> Optional[GameObject]:
    """Cari diamond terdekat dari home"""
    if not papan.diamonds:
        return None

    home = properti.base
    sisa_ruang = properti.inventory_size - properti.diamonds

    terdekat = None
    jarak_terkecil = float('inf')

    for diamond in papan.diamonds:
        if sisa_ruang == 1 and self._adalah_diamond_merah(diamond):
```

```
            continue

        jarak = abs(diamond.position.x - home.x) +
abs(diamond.position.y - home.y)

        if jarak < jarak_terkecil:
            jarak_terkecil = jarak
            terdekat = diamond

    return terdekat
```

3. Metode Pencarian Diamond dari Posisi Saat Ini

```
def _cari_diamond_terdekat_dari_posisi(self, papan: Board, properti,
posisi: Position) -> Optional[GameObject]:
    """Cari diamond terdekat dari posisi saat ini"""
    if not papan.diamonds:
        return None

    sisa_ruang = properti.inventory_size - properti.diamonds

    terdekat = None
    jarak_terkecil = float('inf')

    for diamond in papan.diamonds:
        if sisa_ruang == 1 and self._adalah_diamond_merah(diamond):
            continue

        jarak = abs(diamond.position.x - posisi.x) +
abs(diamond.position.y - posisi.y)

        if jarak < jarak_terkecil:
            jarak_terkecil = jarak
            terdekat = diamond

    return terdekat
```

4. **Metode Pergerakan**

```
   def _bergerak_menuju(self, sekarang: Position, tujuan: Position,
papan: Board) -> Tuple[int, int]:
    """Bergerak langsung menuju target dengan jalur terpendek"""
    if position_equals(sekarang, tujuan):
        return (0, 0)

    dx = tujuan.x - sekarang.x
    dy = tujuan.y - sekarang.y

    # Bergerak ke arah yang lebih jauh terlebih dahulu
    if abs(dx) >= abs(dy):
        langkah = (1 if dx > 0 else -1, 0)
        if self._langkah_valid(sekarang, langkah, papan):
            return langkah
        if dy != 0:
            langkah = (0, 1 if dy > 0 else -1)
            if self._langkah_valid(sekarang, langkah, papan):
                return langkah
    else:
        langkah = (0, 1 if dy > 0 else -1)
        if self._langkah_valid(sekarang, langkah, papan):
            return langkah
        if dx != 0:
            langkah = (1 if dx > 0 else -1, 0)
            if self._langkah_valid(sekarang, langkah, papan):
                return langkah

    # Coba semua arah jika terhalang
    for langkah in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
        if self._langkah_valid(sekarang, langkah, papan):
            return langkah
```

```
    return (0, 0)
```

5. **Penjelasan Alur Program**

   Algoritma BotGreedy mengimplementasikan strategi greedy dengan mode switching untuk mengumpulkan diamond secara efisien. Berikut adalah penjelasan alur program:

   1. **Inisialisasi:**
      - hitung_macet: Menghitung berapa kali bot berada di posisi yang sama secara berturut-turut.
      - posisi_terakhir: Menyimpan posisi bot pada langkah sebelumnya.
      - mode_kumpul: Menentukan strategi pencarian diamond (dari home atau dari posisi saat ini).
   2. **Deteksi Kondisi Macet:**
      - Jika posisi saat ini sama dengan posisi terakhir, increment hitung_macet.
      - Jika berbeda, reset hitung_macet ke 0.
      - Jika hitung_macet ≥ 3, bot mencoba keluar dari kondisi macet dengan mencari langkah valid ke arah lain.
   3. **Pengecekan Inventory:**
      - Jika inventory penuh, bot kembali ke base dan reset mode_kumpul ke false.
   4. **Penentuan Target Diamond:**
      - Jika mode_kumpul = true (mode position-based):
        - Cari diamond terdekat dari posisi saat ini.
        - Jika tidak ditemukan, beralih ke mode home-based.
      - Jika mode_kumpul = false (mode home-based):
        - Cari diamond terdekat dari home base.
        - Jika ditemukan, beralih ke mode position-based.
   5. **Pergerakan ke Target:**
      - Jika target diamond ditemukan, bergerak menuju target tersebut.

- Pergerakan mengutamakan arah dengan jarak terbesar (horizontal atau vertikal).
- Jika arah utama terhalang, coba arah alternatif.

6. **Penanganan Kondisi Khusus:**
    - Jika tidak ada diamond di board, cari dan gunakan red button.
    - Jika masih membawa diamond tetapi tidak ada diamond lagi di board, kembali ke base.
    - Jika semua kondisi di atas tidak terpenuhi, bot tetap di tempat (0, 0).
7. **Validasi Langkah:**
    - - Setiap langkah divalidasi untuk memastikan tidak keluar dari batas board dan tidak bertabrakan dengan bot lain.
8. **Penanganan Diamond Merah:**
    - Jika sisa ruang inventory hanya 1, bot menghindari diamond merah untuk mencegah pemborosan kapasitas.

Alur program ini menunjukkan implementasi strategi greedy yang adaptif, di mana bot selalu memilih diamond terdekat (baik dari home atau posisi saat ini) dan hanya kembali ke base ketika inventory penuh.

4.2 Struktur Data yang Digunakan

Implementasi BotGreedy menggunakan beberapa struktur data utama:

1. Position:

- Struktur data yang merepresentasikan posisi pada grid 2D.
- Memiliki atribut x dan y yang menunjukkan koordinat.
- Digunakan untuk merepresentasikan posisi bot, diamond, base, dan objek lain di board.

2. GameObject:

- Struktur data yang merepresentasikan objek dalam permainan.

- Memiliki atribut position (tipe Position) dan properties.
- Digunakan untuk merepresentasikan bot, diamond, red button, dan objek lain.

3. Board:

- Struktur data yang merepresentasikan keseluruhan papan permainan.
- Memiliki atribut width, height, diamonds (list GameObject), bots (list GameObject), dan game_objects (list GameObject).
- Digunakan untuk mengakses informasi tentang objek di papan permainan.

4. Variabel Status:

- hitung_macet: Integer yang menghitung berapa kali bot berada di posisi yang sama.
- posisi_terakhir: Position yang menyimpan posisi bot pada langkah sebelumnya.
- mode_kumpul: Boolean yang menentukan strategi pencarian diamond.

5. Tuple[int, int]:

- Digunakan untuk merepresentasikan langkah (delta_x, delta_y).
- Nilai yang mungkin: (0, 1), (1, 0), (0, -1), (-1, 0), atau (0, 0).

Kompleksitas ruang dari struktur data yang digunakan adalah O(1) untuk variabel status bot dan O(n + m) untuk akses ke objek di board, di mana n adalah jumlah diamond dan m adalah jumlah bot.

4.3 Pengujian Program

1. **Skenario Pengujian**

   Pengujian BotGreedy dilakukan dengan beberapa skenario untuk mengevaluasi performa dan efektivitasnya:

   Skenario 1: Pengujian Dasar

- Deskripsi: Bot dijalankan pada board standar dengan 4 bot (termasuk BotGreedy) selama 60 detik.
- Tujuan: Mengevaluasi performa dasar dan kemampuan mengumpulkan diamond.

Skenario 2: Pengujian Mode Switching

- Deskripsi: Bot dijalankan pada board dengan diamond yang tersebar dalam cluster-cluster.
- Tujuan: Mengevaluasi efektivitas strategi mode switching dalam mengumpulkan cluster diamond.

2. Hasil Pengujian dan Analisis

Skenario 1: Pengujian Dasar

Hasil:

- Skor Akhir: 14 poin
- Diamond Terkumpul: 12 (10 biru, 2 merah)
- Peringkat: 1 dari 4 bot

Analisis:

- BotGreedy menunjukkan performa yang baik dalam mengumpulkan diamond, dengan fokus pada efisiensi perjalanan.
- Strategi kembali ke base hanya ketika inventory penuh terbukti efektif dalam memaksimalkan skor.
- Bot berhasil mengumpulkan campuran diamond merah dan biru, menunjukkan kemampuan adaptasi terhadap berbagai jenis diamond

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian yang telah dilakukan terhadap BotGreedy dengan strategi Greedy Mode Switching, dapat disimpulkan bahwa:

1. Strategi greedy dengan mode switching efektif dalam mengatasi variasi distribusi diamond di papan permainan. Dengan kemampuan untuk berganti antara mode pencarian berdasarkan posisi base dan posisi saat ini, bot mampu menyesuaikan diri dengan kondisi permainan yang dinamis.

2. Efisiensi pergerakan bot meningkat signifikan dengan hanya kembali ke base saat inventory penuh, sehingga mengurangi waktu tempuh yang tidak produktif.

3. Mekanisme anti-stuck berhasil mengurangi kemungkinan bot terjebak di satu titik dalam jangka waktu lama, dengan tingkat keberhasilan lebih dari 90%.

4. Penanganan khusus terhadap diamond merah, khususnya saat inventory hampir penuh, meningkatkan efisiensi penggunaan kapasitas penyimpanan dan mencegah pemborosan slot.

5. Penggunaan red button secara proaktif saat jumlah diamond sedikit atau habis memastikan bot tetap produktif dan tidak idle, menjaga kontinuitas pengumpulan poin

Secara keseluruhan, strategi greedy yang diimplementasikan dalam BotGreedy berhasil mencapai tujuan lokal secara optimal dalam setiap langkahnya, serta memberikan hasil performa yang kompetitif dibandingkan bot lain dalam berbagai skenario pengujian.

5.2 Saran

Beberapa saran pengembangan lebih lanjut terhadap BotGreedy agar lebih optimal dan adaptif ke depannya antara lain:

1. Integrasi dengan algoritma A* atau Dijkstra untuk menentukan jalur terpendek dengan hambatan (misalnya, saat papan padat oleh bot lain atau obstacle).

2. Pemanfaatan teleporter secara cerdas dalam pengambilan keputusan, dengan mempertimbangkan apakah penggunaan teleport dapat mempercepat perjalanan ke diamond atau base.

3. Penerapan pembelajaran adaptif (misalnya reinforcement learning) agar bot dapat belajar dari pola pergerakan lawan dan kondisi dinamis board untuk strategi jangka panjang.

4. Pengelompokan diamond secara spasial untuk mengenali cluster dan mengarahkan bot ke area dengan kepadatan tinggi diamond.

5. Penyesuaian agresivitas strategi berdasarkan waktu yang tersisa atau posisi skor saat ini, misalnya menjadi lebih ofensif menjelang akhir permainan.

6. Visualisasi dan logging lebih mendetail, seperti heatmap pergerakan bot, waktu idle, dan statistik per langkah, untuk membantu debugging dan analisis strategi.

Dengan pengembangan lebih lanjut, BotGreedy tidak hanya akan unggul dalam efisiensi lokal, tetapi juga dapat beradaptasi lebih baik terhadap permainan yang lebih kompleks dan kompetitif.

LAMPIRAN

A. **Repository Github (https://github.com/ebenlimbong/greedy_diamond)**

B. **Video Penjelasan (https://youtu.be/3VtTYmotkuM)**

DAFTAR PUSTAKA

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

[2] J. Kleinberg dan É. Tardos, Algorithm Design, 1st ed. Boston, MA: Pearson, 2005.

[3] S. Russell dan P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2020.

[4] GeeksforGeeks, "Greedy Algorithm," [Online]. Tersedia: https://www.geeksforgeeks.org/greedy-algorithms/ [Diakses: 1 Juni 2025].

[5] Brilliant.org, "Greedy Algorithms," [Online]. Tersedia: https://brilliant.org/wiki/greedy-algorithm/ [Diakses: 1 Juni 2025].

[6] Python Software Foundation, "The Python Language Reference," [Online]. Tersedia: https://docs.python.org/3/reference/ [Diakses: 1 Juni 2025].

[7] Etimo, "Etimo Game Platform – GitHub Repository," [Online]. Tersedia: https://github.com/etimo/game [Diakses: 1 Juni 2025].