Pemanfaatan Algoritma Greedy Dalam Pembuatan Bot Permainan Diamonds

Tugas Besar

Oleh: Kelompok 10 (POPO SIROYOOO)

Giovan Lado	123140068
Nadine Aura Rahmadhani	123140195
Tengku Hafid Diraputra	123140043

Dosen Pengampu: Winda Yulita, M.Cs.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB 1
DESKRIPSI TUGAS

- Buatlah program sederhana dalam bahasa Python yang mengimplementasikan algoritma Greedy pada bot permainan Diamonds dengan tujuan memenangkan Permainan.
- Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.
- Strategi greedy yang diimplementasikan setiap kelompok harus dikaitkan dengan fungsi objektif dari permainan ini, yaitu memenangkan permainan dengan memperoleh diamond sebanyak banyak nya dan jangan sampai diamond tersebut diambil oleh bot lain. Buatlah strategi greedy terbaik, karena setiap bot dari masing-masing kelompok akan diadu dalam kompetisi Tubes 1.
- Strategi greedy yang kelompok anda buat harus dijelaskan dan ditulis secara eksplisit pada laporan, karena akan diperiksa saat demo apakah strategi yang dituliskan sesuai dengan yang diimplementasikan. Tiap kelompok dapat menggunakan kreativitas yang bermacam macam dalam menyusun strategi greedy untuk memenangkan permainan. Implementasi pemain harus dapat dijalankan pada game engine yang telah disebutkan diatas serta dapat dikompetisikan dengan bot dari kelompok lain.
- Program harus mengandung komentar yang jelas, dan untuk setiap strategi greedy yang disebutkan, harus dilengkapi dengan kode sumber yang dibuat.
- Mahasiswa dilarang menggunakan kode program yang diunduh dari Internet. Mahasiswa harus membuat program sendiri, diperbolehkan untuk belajar dari program yang sudah ada.
- Mahasiswa dianggap sudah melihat dokumentasi dari game engine, sehingga tidak terjadi kesalahpahaman spesifikasi antara mahasiswa dan asisten.
- BONUS (maks 10): Membuat video tentang aplikasi greedy pada bot serta simulasinya pada game kemudian mengunggahnya di Youtube. Video dibuat harus memiliki audio dan menampilkan wajah dari setiap anggota kelompok. Untuk contoh video tubes stima tahun-tahun sebelumnya dapat dilihat di Youtube dengan kata kunci “Tubes Stima”, “strategi algoritma”, “Tugas besar stima”, dll.

- Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional). Dengan catatan asistensi hanya bersifat membimbing, bukan memberikan "jawaban".
- Terdapat juga demo dari program yang telah dibuat. Pengumuman tentang demo menunggu pemberitahuan lebih lanjut dari asisten.
- Bot yang telah dibuat akan dikompetisikan dengan kelompok lain dan disaksikan oleh seluruh peserta kuliah. Terdapat hadiah menarik bagi kelompok yang memenangkan kompetisi.
- Setiap kelompok harap melaporkan nama kelompok dan anggotanya dan diserahkan kepada asisten untuk didata.
- Kelompok yang terindikasi melakukan kecurangan akan diberikan nilai 0 pada tugas Besar.
- Program disimpan dalam repository yang bernama Tubes1_NamaKelompok dengan nama kelompok. Berikut merupakan struktur dari isi repository tersebut:
    A. Folder src berisi source code.
    B. Folder doc berisi laporan tugas besar dengan format NamaKelompok.pdf
    C. README untuk tata cara penggunaan yang minimal berisi:
        I. Penjelasan singkat algoritma greedy yang diimplementasikan
        II. Requirement program dan instalasi tertentu bila ada
        III. Command atau langkah-langkah dalam meng-compile atau build
        IV. program
        V. Author (identitas pembuat)

\

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy merupakan salah satu paradigma pemrograman yang cukup populer dalam menyelesaikan masalah optimasi. Pendekatan yang digunakan di dalam algoritma greedy adalah membuat pilihan yang dapat memberikan perolehan yang terbaik yaitu dengan membuat pilihan minimum pada setiap langkah dengan tujuan sisanya mengarah ke solusi maksimum yang optimal[1]. Algoritma ini bekerja dengan prinsip sederhana namun efektif, yaitu selalu memilih solusi terbaik pada setiap langkah tanpa mempertimbangkan dampak jangka panjang dari keputusan tersebut. Greedy (serakah) disini mengacu pada sifat dari algoritma mengambil pilihan yang paling menguntungkan saat ini juga. Hal ini membuat algoritma greedy tidak selalu menghasilkan solusi optimal keseluruhan tapi hanya solusi lokal dan kombinasi dari semua langkah tersebut belum tentu menghasilkan hasil terbaik keseluruhan.

2.2 Cara Kerja Program

Program ini dirancang sebagai bot permainan Diamonds untuk berjalan secara otomatis dengan menjalankan keputusan cerdas pada setiap langkah berdasarkan kondisi terkini papan permainan. Kegunaan utama dari bot ini adalah implementasi algoritma greedy yang memandu setiap aksi bot, dimulai dari pengambilan diamond hingga kembali ke markas. Bot berinteraksi dengan game engine melalui sebuah script utama yang memproses logika bot dan menerjemahkannya menjadi aksi dalam permainan.

2.2.1 Cara Implementasi Program

Implementasi bot ini berpusat pada kelas MyBot yang diwariskan (inheritance) dari kelas BaseLogic yang disediakan oleh kerangka kerja permainan. Logika pengambilan keputusan bot tertuang dalam metode utama next_move(self, board_bot: GameObject, board: Board) di dalam kelas MyBot.

Pada setiap pemanggilan, metode next_move melakukan langkah-langkah berikut:

- Pengambilan informasi awal: bot mengambil informasi esensial dari parameter yang diterima, yaitu board_bot (representasi objek bot itu sendiri di papan) dan board (representasi keseluruhan papan permainan). Dari sini, didapatkan posisi bot saat ini (current_pos), dimensi papan (board_width, board_height), serta properti-properti penting bot seperti jumlah diamond yang sedang dibawa (current_diamonds_held), kapasitas maksimum inventory (inventory_size), dan

data posisi markas (base_pos_data). Data posisi markas kemudian diproses untuk mendapatkan objek position yang valid.

- Penentuan Target (logika greedy):
    - Prioritas kembali ke markas: Bot akan memeriksa apakah inventorynya penuh (current_diamonds_held $\geq$ inventory_size) dan apakah ia mengetahui lokasi markasnya (base_position) valid. Jika kedua kondisi ini terpenuhi, target utama bot (target_pos) akan diatur ke posisi markas. Ini merupakan pilihan pertama karena mengamankan diamond yang sudah dikumpulkan adalah prioritas jika tidak bisa membawa lebih banyak.
    - Prioritas pengumpulan diamond: Jika bot belum kembali ke markas (inventory masih tercukupi), maka akan beralih ke mode pengumpulan diamond. Bot akan mengidentifikasi semua RedDiamondGameObject dan DiamondGameObject di papan. Kemudian, menggunakan fungsi pembantu find_closest diamond, bot mencari posisi diamond merah terdekat dan diamond biru terdekat.
    - Keputusan greedy berikutnya adalah membandingkan jarak antara kedua jenis diamond. Jika diamond merah dengan jaraknya lebih dekat atau sama dengan jarak diamond biru terdekat, maka diamond merah tersebut yang menjadi target pengambilan. Logika bertujuan mendapatkan poin sebanyak mungkin.
- Perhitungan langkah: Setelah target_pos ditentukan, fungsi pembantu _move_towards(current_pos, target_pos) dipanggil. Fungsi ini secara greedy menghitung satu langkah (delta_x, delta_y) yang paling langsung mengurangi jarak ke target, dengan memprioritaskan pergerakan pada sumbu X terlebih dahulu, kemudian sumbu Y.
- Validasi dan gerakan acak (fallback): langkah yang dihitung kemudian divalidasi untuk memastikan bot tidak bergerak keluar dari batas papan. Jika tidak valid atau tidak ada target strategis, gerakan di reset (delta_x=0, delta_y=0). Dalam kondisi ini, bot memanggil get_valid_moves untuk mendapatkan semua langkah valid dari posisinya, lalu memilih satu secara acak. Ini memastikan bot tetap bergerak jika tidak ada pilihan greedy yang jelas.

2.2.2 Menjalankan Bot Program

Untuk menjalankan bot ini, pengguna perlu menjalankan skrip utama dari permainan biasanya berupa file bernama *main.py* atau *mybot.py* . Proses ini dilakukan melalui terminal atau command prompt.

Langkah-langkahnya secara umum adalah sebagai berikut:

1. Buka terminal atau *command prompt* (menggunakan dua cmd).
2. Navigasi ke direktori utama proyek permainan (cmd1 misalnya, cd tubes1-IF2211-game-engine-1.1.0 dan cmd 2, cd tubes1-IF2211-bot-starter-pack.1.0.1).
3. Jalankan bot dengan melakukan running di cmd1 dengan *npm run start* dan cmd2 dengan *run-bots.bat atau python main.py --logic MyBot --email=bot1@email.com --name=pemain1 --password=123456 --team etimo* jika ingin satu bot saja yang dijalankan.

Saat dijalankan, *main.py* akan mengatur konfigurasi, menyiapkan papan permainan, dan membuat objek *MyBot* (dengan logika di *game/logic/mybot.py*). Game engine lalu memanggil *next_move* dari bot di setiap gilirannya, menjalankan strategi greedy secara otomatis. Umumnya, pengguna hanya bertindak sebagai pengamat, kecuali jika framework game menyediakan fitur interaktif tambahan.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Untuk menerapkan algoritma greedy, masalah dalam permainan *Diamonds* perlu dipetakan ke dalam elemen-elemen yang dapat dievaluasi untuk pengambilan keputusan "serakah" pada setiap langkah. Proses mapping ini melibatkan identifikasi:

- Keadaan saat ini (current state):
    - Posisi bot saat ini (current_pos): direpresentasikan sebagai objek Position (x, y).
    - Status inventory bot: jumlah diamond yang dibawa (current_diamonds_held) dan kapasitas maksimum inventory (inventory_size).
    - Posisi markas (base_position): lokasi(x, y) tempat bot dapat menyimpan diamond.
- Objek-objek penting di papan:
    - *Diamonds*: dibedakan menjadi RedDiamondGameObject dan DiamondGameObject (diasumsikan sebagai diamond biru/standar). Masing-masing memiliki posisi.
    - Papan permainan: Dimensi papan (board_width, board_height) yang membatasi pergerakan.
- Aksi yang mungkin:
    - Bergerak satu petak ke atas, bawah, kiri, atau kanan (delta_x, delta_y).
- Tujuan Greedy:
    - Jangka pendek: bergerak ke petak yang paling cepat mendekatkan bot ke objek bernilai (diamond atau markas saat penuh).
    - Jangka panjang: mengumpulkan skor setinggi mungkin.

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam perancangan strategi bot untuk permainan diamond collector ini, beberapa alternatif algoritma berbasis metode Greedy dipertimbangkan sebelum menentukan solusi akhir yang diterapkan. Adapun alternatif strategi yang dipikirkan adalah sebagai berikut:

1. Greedy Berdasarkan Jarak Terdekat (Nearest Diamond)

Strategi ini memilih diamond dengan jarak paling dekat dari posisi bot saat ini. Jarak dihitung menggunakan metode Manhattan Distance. Strategi ini diprioritaskan karena paling sederhana, cepat, dan sesuai dengan sifat permainan yang dinamis.

2. Greedy Berdasarkan Nilai Diamond

   Alternatif ini mempertimbangkan nilai dari setiap diamond, di mana diamond dengan nilai tertinggi menjadi prioritas. Akan tetapi, dalam permainan ini seluruh diamond memiliki nilai yang sama, sehingga strategi ini tidak diterapkan.

3. Greedy Competitive Distance & Base Return

   Strategi ini mengutamakan efisiensi pergerakan dengan memilih diamond terdekat sebagai target utama dan secara otomatis kembali ke markas saat kapasitas penyimpanan penuh. Pendekatan ini dirancang untuk mempercepat siklus pengumpulan dan meminimalkan waktu yang terbuang. Selain itu, ketika tidak ada target yang tersedia di sekitar, bot tetap bergerak secara acak untuk mencegah kondisi diam yang tidak produktif. Strategi ini terbukti efektif dalam menjaga alur permainan tetap dinamis dan meningkatkan potensi skor secara konsisten.

4. Greedy Random Move

   Sebagai alternatif pembanding, dipertimbangkan pula strategi di mana bot bergerak secara acak tanpa mempertimbangkan posisi diamond maupun jarak. Strategi ini hanya berfungsi sebagai baseline untuk mengukur efektivitas strategi lainnya

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setelah melakukan eksplorasi terhadap beberapa alternatif solusi strategi Greedy pada sub bab sebelumnya, dilakukan analisis terhadap masing-masing alternatif untuk menilai efisiensi dan efektivitasnya dalam konteks permainan diamond collector ini. Berikut adalah hasil analisis terhadap keempat alternatif strategi tersebut:

Alternatif strategi	Kelebihan	Kekurangan
Greedy Berdasarkan Jarak Terdekat	● Cepat dalam menentukan tujuan karena hanya menghitung jarak. ● Implementasi sederhana. ● Efektif di lingkungan permainan dengan nilai	● Tidak mempertimbangkan kemungkinan blokade dari bot lawan. ● Hanya mengutamakan jarak, tanpa analisis potensi ancaman.

	item seragam.	
Greedy Berdasarkan Nilai Diamond	● Memprioritaskan diamond bernilai tinggi untuk skor maksimal.	● Tidak relevan di permainan ini karena seluruh diamond memiliki nilai yang sama.
Greedy Competitive Distance & Base Return	● Menjaga dinamika permainan ● Sederhana untuk diimplementasikan ● Pengamanan skor otomatis	● Rentan terhadap "perangkap lokal" ● Mengabaikan rintangan
Greedy Random Move	● Implementasi sangat sederhana. ● Tidak memerlukan perhitungan jarak.	● Sangat tidak efisien. ● Performa buruk karena pergerakan acak yang tidak terarah.

Dari tabel di atas dapat disimpulkan bahwa Greedy Berdasarkan Jarak Terdekat merupakan strategi yang paling sesuai dengan kebutuhan permainan ini. Strategi ini mampu memberikan hasil optimal dengan proses perhitungan sederhana dan cepat tanpa memerlukan data tambahan di luar posisi diamond.

3.4 Strategi Greedy yang Dipilih

Berdasarkan hasil eksplorasi dan analisis terhadap keempat alternatif strategi Greedy yang telah dipaparkan pada subbab sebelumnya, maka strategi yang dipilih untuk diimplementasikan dalam program adalah Greedy Competitive Distance & Base Return. Strategi ini dipilih karena menawarkan keseimbangan terbaik antara kesederhanaan implementasi, efisiensi pergerakan, dan efektivitas dalam pengumpulan skor.

Berikut adalah rincian strategi yang dipilih dan diimplementasikan:

1. Prioritas utama: kembali ke markas jika penuh.
    - Pada setiap giliran, bot pertama kali memerikasa:
        - Apakah lokasi markas (base_position) diketahui
        - Apakah jumlah diamond yang dibawa (current_diamonds_held) sudah mencapai atau melebihi kapasitas inventory (inventory_size)

- Jika kedua kondisi terpenuhi, bot akan menetapkan posisi markas sebagai target (target_pos). Ini memastikan diamond yang sudah dikumpulkan bisa diamankan dan bot bisa mengambil diamond baru.

2. Prioritas kedua: Mencari dan mengumpulkan diamond (jika tidak kembali ke markas).

- Jika bot tidak perlu kembali ke markas, ia akan mencari diamond.
- Bot mengidentifikasi semua RedDiamondGameObject dan DiamondGameObject yang ada di papan.
- Bot kemudian mencari posisi diamond merah terdekat (closest_red_pos) dan diamond biru (closesr_blue_pos) menggunakan fungsi find_closest_diamond.
- Keputusan greedy berdasarkan jarak kompetitif:
    - Jika ada diamond merah (closest_red_pos) ditemukan:
        - Bot menghitung jarak ke diamond merah tersebut.
        - Jika ada diamond biru (closest_blue_pos) ditemukan:
            - Bot menghitung jarak ke diamond biru tersebut.
            - Jika jarak ke diamond merah lebih kecil atau sama dengan jarak ke diamond biru, maka target_pos adalah closest_red_pos.
            - Jika tidak, target_pos adalah closest_blue_pos.
        - Jika tidak ada diamond biru (hanya merah yang ada), maka target_pos adalah closest_red_pos.
    - Jika tidak ada diamond merah tetapi ada diamond biru, maka target_pos adalah closest_blue_pos.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

```
// Konstanta
CONSTANT DEFAULT_MAX_DIAMONDS_HELD = 5

// Fungsi Utama: Menentukan langkah bot berikutnya
FUNCTION next_move(bot_object, board_info):
    current_pos = bot_object.position
    delta_x = 0
    delta_y = 0

    board_width = board_info.width
    board_height = board_info.height

    // Ambil properti bot (jumlah diamond, kapasitas, posisi base)
    current_diamonds_held = 0
    IF bot_object.properties HAS ATTRIBUTE "diamonds" THEN
        current_diamonds_held = bot_object.properties.diamonds
    END IF

    inventory_size = DEFAULT_MAX_DIAMONDS_HELD
    IF bot_object.properties HAS ATTRIBUTE "inventory_size" THEN
        inventory_size = bot_object.properties.inventory_size
    END IF

    base_pos_data = NULL
    IF bot_object.properties HAS ATTRIBUTE "base" THEN
        base_pos_data = bot_object.properties.base
    END IF

    base_position_object = NULL
```

```
    IF base_pos_data IS INSTANCE OF PositionClass THEN
        base_position_object = base_pos_data
    ELSE IF base_pos_data IS A DICTIONARY AND HAS KEYS "x" AND "y"
THEN
        base_position_object = CREATE_POSITION(base_pos_data["x"],
base_pos_data["y"])
    ELSE IF base_pos_data IS A LIST OR TUPLE WITH 2 ELEMENTS THEN
        base_position_object = CREATE_POSITION(base_pos_data[0],
base_pos_data[1])
    END IF

    target_destination_pos = NULL

    // Logika Greedy 1: Kembali ke Markas Jika Penuh
    IF base_position_object IS NOT NULL AND current_diamonds_held >=
inventory_size THEN
        target_destination_pos = base_position_object
    ELSE
        // Logika Greedy 2: Kumpulkan Diamond
        all_game_entities = board_info.game_objects
        red_diamond_list = EMPTY_LIST
        FOR EACH entity IN all_game_entities:
            IF entity.type == "RedDiamondGameObject" THEN
                ADD entity TO red_diamond_list
            END IF
        END FOR

        blue_diamond_list = EMPTY_LIST
        FOR EACH entity IN all_game_entities:
            IF entity.type == "DiamondGameObject" THEN
                ADD entity TO blue_diamond_list
            END IF
        END FOR

        closest_red_diamond_pos =
find_closest_object_logic(current_pos, red_diamond_list, board_info)
```

```
        closest_blue_diamond_pos =
find_closest_object_logic(current_pos, blue_diamond_list,
board_info)

        // Prioritaskan diamond merah jika lebih dekat atau sama
jaraknya
        IF closest_red_diamond_pos IS NOT NULL THEN
            distance_to_red =
calculate_manhattan_distance(current_pos, closest_red_diamond_pos)
            IF closest_blue_diamond_pos IS NOT NULL THEN
                distance_to_blue =
calculate_manhattan_distance(current_pos, closest_blue_diamond_pos)
                IF distance_to_red <= distance_to_blue THEN
                    target_destination_pos = closest_red_diamond_pos
                ELSE
                    target_destination_pos =
closest_blue_diamond_pos
                END IF
            ELSE // Hanya ada diamond merah
                target_destination_pos = closest_red_diamond_pos
            END IF
        ELSE IF closest_blue_diamond_pos IS NOT NULL THEN // Tidak
ada merah, tapi ada biru
            target_destination_pos = closest_blue_diamond_pos
        END IF
    END IF

    // Pergerakan Menuju Target
    IF target_destination_pos IS NOT NULL THEN
        delta_x, delta_y = move_towards_target_logic(current_pos,
target_destination_pos)
    END IF

    // Validasi Gerakan & Fallback Gerakan Acak
    next_potential_x = current_pos.x + delta_x
    next_potential_y = current_pos.y + delta_y
```

```
    IF is_out_of_bounds(next_potential_x, next_potential_y,
board_width, board_height) THEN
        delta_x = 0
        delta_y = 0
    END IF

    IF delta_x == 0 AND delta_y == 0 THEN
        list_of_valid_random_moves =
get_valid_random_moves_logic(current_pos, board_width, board_height)
        IF list_of_valid_random_moves IS NOT EMPTY THEN
            delta_x, delta_y =
CHOOSE_RANDOM_FROM(list_of_valid_random_moves)
        END IF
    END IF

    RETURN delta_x, delta_y
END FUNCTION

// --- Pseudocode Fungsi Pembantu ---
FUNCTION find_closest_object_logic(bot_current_pos, list_of_objects,
board_info_param): // board_info_param ditambahkan jika diperlukan
oleh logika internal, jika tidak bisa dihilangkan
    closest_found_object_pos = NULL
    smallest_distance_so_far = INFINITY

    FOR EACH object_item IN list_of_objects:
        current_distance =
calculate_manhattan_distance(bot_current_pos, object_item.position)
        IF current_distance < smallest_distance_so_far THEN
            smallest_distance_so_far = current_distance
            closest_found_object_pos = object_item.position
        END IF
    END FOR
    RETURN closest_found_object_pos
END FUNCTION
```

```
FUNCTION get_valid_random_moves_logic(bot_current_pos,
game_board_width, game_board_height):
    list_to_store_valid_moves = EMPTY_LIST
    possible_move_deltas = [(0,1), (0,-1), (1,0), (-1,0)]

    FOR EACH (dx_candidate, dy_candidate) IN possible_move_deltas:
        next_potential_x = bot_current_pos.x + dx_candidate
        next_potential_y = bot_current_pos.y + dy_candidate
        IF is_within_bounds(next_potential_x, next_potential_y,
game_board_width, game_board_height) THEN
            ADD (dx_candidate, dy_candidate) TO
list_to_store_valid_moves
        END IF
    END FOR
    RETURN list_to_store_valid_moves
END FUNCTION

FUNCTION move_towards_target_logic(current_bot_pos,
target_destination_pos):
    calculated_delta_x = 0
    calculated_delta_y = 0

    IF target_destination_pos.x > current_bot_pos.x THEN
        calculated_delta_x = 1
    ELSE IF target_destination_pos.x < current_bot_pos.x THEN
        calculated_delta_x = -1
    END IF

    IF calculated_delta_x == 0 THEN
        IF target_destination_pos.y > current_bot_pos.y THEN
            calculated_delta_y = 1
        ELSE IF target_destination_pos.y < current_bot_pos.y THEN
            calculated_delta_y = -1
        END IF
    END IF
```

```
    RETURN calculated_delta_x, calculated_delta_y
END FUNCTION

// Fungsi utilitas (diasumsikan ada)
// FUNCTION calculate_manhattan_distance(pos1, pos2)
// FUNCTION is_out_of_bounds(x, y, width, height)
// FUNCTION is_within_bounds(x, y, width, height)
// FUNCTION CREATE_POSITION(x,y)
```

4.1.2 Penjelasan Alur Program:

Alur program utama bot berada dalam metode next_move pada kelas MyBot. Setiap kali metode ini dipanggil oleh *game engine*, serangkaian langkah logis dieksekusi untuk menentukan aksi bot selanjutnya:

1. Inisialisasi dan pengambilan data: program menginisialisasi variabel untuk pergerakan (delta_x, delta_y) dan mengambil informasi penting seperti posisi bot saat ini, dimensi papan, jumlah diamond yang dibawa, kapasitas inventory, dan data posisi markas. Data posisi markas kemudian dikonversi menjadi format Position yang konsisten.
2. Keputusan Kembali ke Markas: Bot pertama-tama mengevaluasi apakah inventorynya penuh dan apakah lokasi markas diketahui. Jika ya, target pergerakan bot (target_destination_pos) diatur ke posisi markas. Ini merupakan pilihan greedy pertama untuk mengamankan skor.
3. Keputusan Mencari Diamond: Jika bot tidak perlu kembali ke markas, ia akan mencari diamond. Program memfilter objek di papan untuk mendapatkan daftar diamond merah dan diamond biru. Menggunakan fungsi find_closest_diamond (direpresentasikan sebagai find_closest_object_logic dalam pseudocode), bot menemukan diamond merah terdekat dan diamond biru terdekat.
4. Prioritise Diamond: Bot membandingkan jarak ke diamond merah dan biru terdekat. Jika diamond merah ada dan jaraknya lebih dekat atau sama dengan diamond biru (atau jika hanya diamond merah yang ada), diamond merah tersebut menjadi target. Jika tidak, diamond biru terdekat yang menjadi target. Ini adalah pilihan greedy kedua yang bertujuan mengoptimalkan pengumpulan berdasarkan kedekatan dan potensi nilai.

5. Perhitungan Langkah Menuju Target: Setelah target ditentukan (markas atau diamond), fungsi _move_towards (direpresentasikan sebagai move_towards_target_logic) menghitung langkah (delta_x, delta_y) yang paling langsung menuju target. Logika ini secara greedy mencoba mengurangi jarak pada sumbu X terlebih dahulu, kemudian sumbu Y.
6. Validasi Langkah: Langkah yang telah dihitung kemudian divalidasi untuk memastikan bot tidak bergerak keluar dari batas papan permainan. Jika langkah tersebut tidak valid, delta_x dan delta_y direset menjadi 0.
7. Fallback Gerakan Acak: Jika tidak ada target strategis yang ditentukan (misalnya, tidak ada diamond dan inventory tidak penuh) atau jika langkah strategis yang dihitung ternyata tidak valid (menyebabkan delta_x dan delta_y menjadi 0), bot akan mencoba bergerak secara acak. Fungsi get_valid_moves (direpresentasikan sebagai get_valid_random_moves_logic) dipanggil untuk mendapatkan semua kemungkinan gerakan yang valid dari posisi saat ini, dan satu gerakan dipilih secara acak. Ini memastikan bot tetap aktif dan tidak terjebak.
8. Pengembalian Aksi: Akhirnya, metode next_move mengembalikan nilai delta_x dan delta_y yang akan digunakan oleh *game engine* untuk menggerakkan bot.

4.2 Struktur Data yang Digunakan

- Position: Sebuah objek atau kelas yang digunakan untuk merepresentasikan koordinat (x, y) di papan permainan. Atribut utamanya adalah .x dan.y. Ini digunakan untuk melacak posisi bot, diamond, dan markas.
- GameObject: Kelas dasar untuk semua entitas dalam permainan, termasuk bot (board_bot), diamond, dan rintangan (jika ada). Setiap GameObject memiliki atribut position (sebuah objek Position) dan type (sebuah *string* yang mengidentifikasi jenis objek, misalnya, "RedDiamondGameObject", "DiamondGameObject"). Objek bot (board_bot) juga memiliki atribut properties yang menyimpan data spesifik bot seperti diamonds (jumlah diamond yang dibawa), inventory_size, dan base (data posisi markas).
- Board: Objek yang merepresentasikan keseluruhan papan permainan. Ini berisi daftar semua GameObject yang ada di papan melalui atribut game_objects. Papan juga menyediakan informasi mengenai dimensinya melalui atribut width dan height.
- list (Python List): Digunakan secara ekstensif untuk:
    - Menyimpan kumpulan GameObject (misalnya, hasil filter red_diamonds, blue_diamonds dari board.game_objects).

    - Menyimpan daftar gerakan yang valid yang dikembalikan oleh get_valid_moves, di mana setiap gerakan adalah sebuah tuple.
- tuple (Python Tuple): Digunakan untuk merepresentasikan perubahan posisi (delta_x, delta_y) yang dikembalikan oleh next_move dan fungsi-fungsi pergerakan lainnya. Juga digunakan dalam get_valid_moves untuk menyimpan pasangan (dx, dy) dari kemungkinan gerakan.
- float('inf'): Digunakan dalam find_closest_diamond sebagai nilai awal untuk min_distance, memastikan bahwa jarak ke diamond pertama yang diperiksa akan selalu lebih kecil.
- None (Python NoneType): Digunakan untuk mengindikasikan ketiadaan nilai, misalnya jika tidak ada diamond yang ditemukan (closest_diamond_pos = None) atau jika data posisi markas tidak tersedia.

4.3 Pengujian Program

4.3.1 Skenario Pengujian

Berikut adalah beberapa skenario pengujian yang dirancang dan dijalankan:

- Skenario 1: Papan Kosong (Tidak Ada Diamond, Inventory Kosong)
    - *Harapan:* Bot seharusnya bergerak secara acak karena tidak ada target diamond dan tidak perlu kembali ke markas.
- Skenario 2: Satu Diamond Biru di Papan
    - *Harapan:* Bot seharusnya bergerak menuju satu-satunya diamond biru tersebut.
- Skenario 3: Satu Diamond Merah di Papan
    - *Harapan:* Bot seharusnya bergerak menuju satu-satunya diamond merah tersebut.
- Skenario 4: Diamond Merah dan Biru, Merah Lebih Dekat
    - *Harapan:* Bot seharusnya memprioritaskan dan bergerak menuju diamond merah.
- Skenario 5: Diamond Merah dan Biru, Biru Lebih Dekat
    - *Harapan:* Bot seharusnya bergerak menuju diamond biru.
- Skenario 6: Diamond Merah dan Biru, Jarak Sama
    - *Harapan:* Bot seharusnya memprioritaskan dan bergerak menuju diamond merah (sesuai logika dist_red <= dist_blue).
- Skenario 7: Inventory Penuh, Ada Diamond di Dekat, Markas Diketahui
    - *Harapan:* Bot seharusnya mengabaikan diamond dan bergerak menuju markas.

- Skenario 8: Inventory Penuh, Markas Tidak Diketahui
    - *Harapan:* Bot akan terus mencoba mencari diamond (karena kondisi kembali ke markas tidak terpenuhi), atau bergerak acak jika tidak ada diamond. Perilaku ini menyoroti pentingnya data markas yang valid.
- Skenario 9: Bot Terhalang Menuju Target (Simulasi Sederhana dengan Batas Papan)
    - *Harapan:* Jika langkah langsung menuju target keluar batas, bot seharusnya tidak keluar batas dan idealnya melakukan gerakan acak yang valid.
- Skenario 10: Tidak Ada Diamond dan Inventory Tidak Penuh
    - *Harapan:* Bot seharusnya bergerak acak.

4.3.2 Hasil Pengujian dan Analisis

- Pada Skenario 1, 2, dan 3, bot berhasil mengidentifikasi target tunggal (atau ketiadaan target) dan bergerak sesuai harapan (acak atau menuju diamond).
- Pada Skenario 4, 5, dan 6, logika prioritas diamond berdasarkan jenis dan jarak kompetitif berjalan dengan benar, di mana bot memilih diamond merah jika lebih dekat atau sama jaraknya, dan memilih biru jika biru secara signifikan lebih dekat.
- Skenario 7 menunjukkan bahwa logika kembali ke markas saat inventory penuh berfungsi dengan baik, mengesampingkan diamond terdekat.
- Skenario 8 mengonfirmasi bahwa tanpa informasi markas yang valid, bot tidak dapat menjalankan fungsi kembali ke markas, yang sesuai dengan implementasi.
- Pada Skenario 9, mekanisme validasi langkah dan fallback ke gerakan acak mencegah bot keluar dari batas papan, meskipun belum ada logika pathfinding cerdas untuk melewati rintangan.
- Skenario 10 juga menunjukkan bot bergerak acak seperti yang diharapkan.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Hasil implementasi dan pengujian bot permainan Diamonds dengan strategi "Greedy Competitive Distance & Base Return" menunjukkan keberhasilan penerapan algoritma greedy. Bot mampu secara mandiri menentukan target (diamond atau markas) dan bergerak menuju pilihan optimal lokal. Strategi ini, yang memprioritaskan kembali ke markas saat inventaris penuh dan memilih diamond terdekat dengan preferensi pada diamond merah, terbukti cukup efektif dalam berbagai skenario pengujian, memungkinkan bot mengumpulkan diamond dan mengamankan skor secara rasional. Namun, keterbatasan inheren dari algoritma greedy membuat bot tidak selalu mencapai solusi optimal global karena keputusan diambil sesaat tanpa perencanaan jangka panjang, terutama pada peta kompleks atau saat menghadapi rintangan. Fungsi kembali ke markas juga sangat bergantung pada akurasi data posisi markas. Sebagai mekanisme fallback, gerakan acak berhasil mencegah bot diam saat tidak ada target strategis atau jika langkah yang dihitung tidak valid, sehingga bot terus menjelajahi papan.

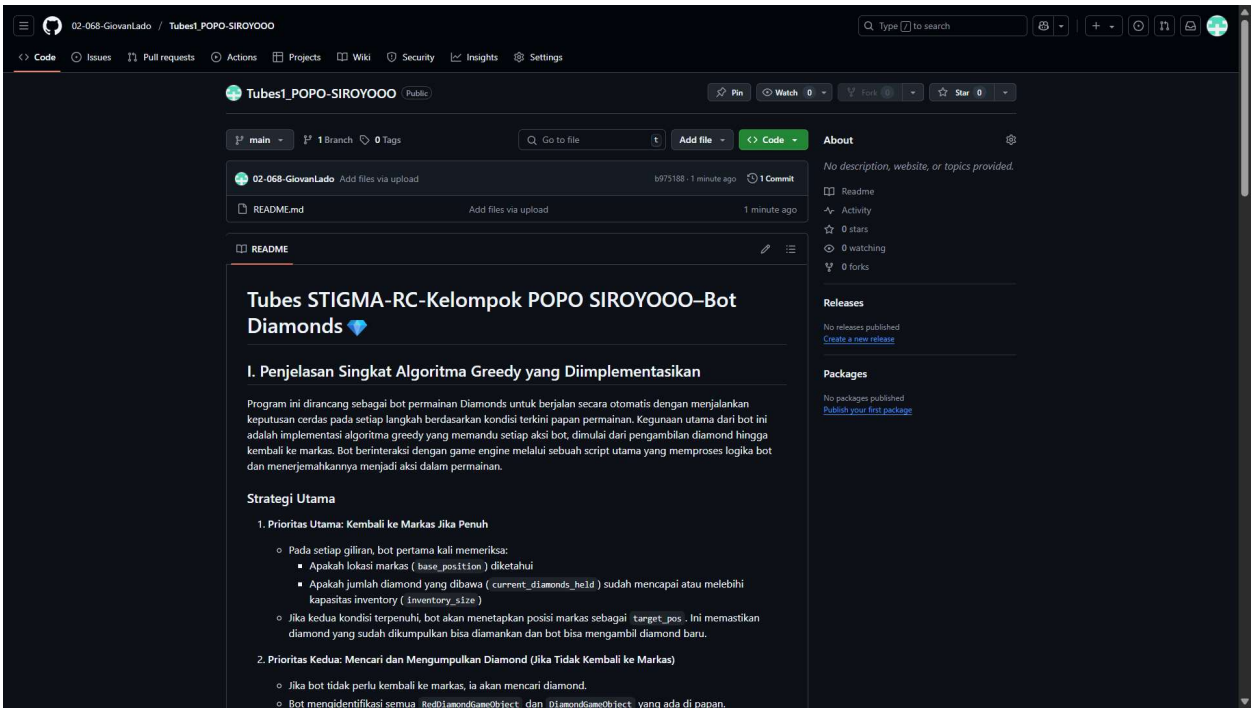
5.2 Saran

Untuk meningkatkan kecerdasan dan daya saing bot Diamonds di masa depan, beberapa saran pengembangan dapat dipertimbangkan, seperti implementasi pathfinding lanjutan (misalnya A* atau Dijkstra) untuk navigasi rintangan yang efektif dan penemuan jalur optimal. Selain itu, strategi dapat disempurnakan dengan pengenalan bobot atau nilai diamond yang lebih dinamis untuk keputusan greedy yang lebih cerdas secara ekonomi, serta mekanisme anti-stagnasi yang lebih baik melalui eksplorasi terstruktur. Jika relevan, kemampuan mendeteksi dan menghindari elemen berbahaya atau lawan, pengembangan strategi adaptif berdasarkan kondisi permainan, dan penggabungan heuristik tambahan seperti preferensi bergerak ke area dengan visibilitas diamond lebih tinggi atau menghindari sudut mati, juga dapat secara signifikan memperkaya pengambilan keputusan bot.

LAMPIRAN

A. Repository Github (link)

Link: https://github.com/02-068-GiovanLado/Tubes1_POPO-SIROYOOO



B. Video Penjelasan (link GDrive)

https://drive.google.com/drive/u/0/folders/1uDrP90qV_jMZsA8sXf76_5WD1fwwJ0cN

DAFTAR PUSTAKA

[1] R. Sabaruddin, “Solusi Optimum Minmax 0/1 Knapsack Menggunakan Algoritma Greedy,” *EvolusiJurnal sains dan Manaj.*, vol. Volume 4 N, no. May, pp. 31–48, 2016, [Online]. Available: https://ejournal.bsi.ac.id/ejurnal/index.php/evolusi/article/view/703/578