IMPLEMENTASI ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN DIAMOND

Tugas Besar

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

Oleh : Kelompok 6 (Sigma)

Margaretta Angela Manullang	123140010
Anselmus Herpin Hasugian	123140020
Mario Fransiskus Sitepu	123140023

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

DAFTAR ISI **1**
BAB I DESKRIPSI TUGAS **3**
BAB II LANDASAN TEORI **8**
2.1 Dasar Teori 8
1. Cara Implementasi Program 9
2. Menjalankan Bot Program 9
BAB III APLIKASI STRATEGI GREEDY **10**
3.1 Proses Mapping 10
3.2 Eksplorasi Alternatif Solusi Greedy 10
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy 11
3.4 Strategi Greedy yang Dipilih 11
BAB IV IMPLEMENTASI DAN PENGUJIAN **12**
4.1 Implementasi Algoritma Greedy 12
1. Pseudocode 12
2. Penjelasan Alur Program 22
A. Inisialisasi 22
B. Fungsi Utama (next_move) 22
C. Algoritma Pencarian Target (find_optimal_target) 22
D. Perhitungan DHG Score 23
E. Sistem Keamanan dan Anti-Stuck 23
F. Strategi Kembali ke Base 23
G. Adaptasi Kompetitif 24
H. Optimasi Lanjutan 24
4.2 Struktur Data yang Digunakan 25
4.3 Pengujian Program 25
1. Skenario Pengujian 25
2. Hasil Pengujian dan Analisis 26
BAB V KESIMPULAN DAN SARAN **27**
5.1 Kesimpulan 27
5.2 Saran 27
LAMPIRAN **28**
DAFTAR PUSTAKA **29**

DAFTAR GAMBAR & TABEL

Gambar 1.1 Layar Permainan Diamonds..... 4
Gambar 1.2 Varian Diamond Biru Gambar 1.3 Varian Diamond Merah..... 5
Gambar 1.4 Red Button / Diamond Button..... 6
Gambar 1.5 Teleporters..... 6
Gambar 1.6 Bot Dengan Name “stima”..... 6
Gambar 1.7 Base Dengan Name “stima”..... 7
Gambar 1.8 Layar Inventory..... 7
Tabel 4.3 Hasil Pengujian dan Analisis.....27

BAB I
DESKRIPSI TUGAS

Diamonds adalah tantangan pemrograman yang menantang para peserta untuk bertanding menggunakan bot yang mereka kembangkan melawan bot peserta lain. Setiap peserta akan memiliki bot dengan tujuan utama mengumpulkan sebanyak mungkin diamond. Namun, tantangan ini tidak akan mudah karena berbagai hambatan akan menambah keseruan dan kerumitan dalam permainan. Untuk memenangkan kompetisi, peserta harus menerapkan strategi khusus pada bot mereka masing-masing. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

Gambar 1.1 Layar Permainan Diamonds

Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan **strategi *greedy*** dalam membuat bot ini.

Program permainan Diamonds terdiri atas :

1. Game engine, yang secara umum berisi :
   a. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot

b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan

2. Bot starter pack, yang secara umum berisi :
   a. Program untuk memanggil API yang tersedia pada backend
   b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)
   c. Program utama (main) dan utilitas lainnya

Komponen-komponen dari permainan Diamonds antara lain :

1. Diamonds

   Untuk meraih kemenangan dalam pertandingan, pemain harus mengumpulkan sejumlah diamond dengan cara melewati atau mengambilnya. Ada dua varian diamond dalam permainan : diamond merah dan biru. Setiap diamond merah memiliki nilai dua poin, sementara diamond biru bernilai satu poin. Diamond akan terus muncul kembali secara periodik, dan proporsi antara diamond merah dan biru akan berubah-ubah setiap kali terjadi regenerasi.

Gambar 1.2 Varian Diamond Biru

Gambar 1.3 Varian Diamond Merah

2. Red Button / Diamond Button

   Saat tombol merah dilewati atau dilangkahi, semua diamond, diamond merah maupun diamond biru, akan muncul kembali di papan permainan dengan posisi yang acak. Lokasi tombol merah itu sendiri juga akan berubah ke posisi acak setelah tombol tersebut diaktifkan.

Gambar 1.4 Red Button / Diamond Button

3. Teleporters

   Terdapat 2 *teleporter* yang saling terhubung satu sama lain. Jika bot melewati sebuah *teleporter* maka bot akan berpindah menuju posisi *teleporter* yang lain.

Gambar 1.5 Teleporters

4. Bots

   Pada game ini kita akan menggerakkan bot untuk mendapatkan *diamond* sebanyak banyaknya. Bot ini bisa men-*tackle* bot lainnya.

Gambar 1.6 Bot Dengan Name “stima”

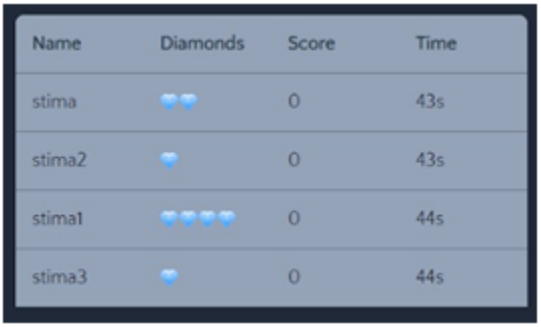
5. Bases

   Setiap bot dilengkapi dengan sebuah Base, yang berfungsi sebagai tempat untuk menampung diamond yang dikumpulkan. Ketika diamond disimpan di Base, skor dari bot tersebut akan meningkat sesuai dengan nilai diamond yang disimpan, dan inventaris bot (yang akan diuraikan lebih lanjut) akan dikosongkan.

Gambar 1.7 Base Dengan Name “stima”

6. Inventory

   Bot dilengkapi dengan sebuah inventaris, yang berperan sebagai lokasi penyimpanan sementara untuk diamond yang telah dikumpulkan. Inventaris ini dibatasi oleh sebuah kapasitas maksimal, yang artinya dapat terisi penuh. Untuk menghindari keadaan penuh tersebut, bot dapat mentransfer isi inventaris mereka ke Base, sehingga memungkinkan inventaris tersebut untuk dikosongkan kembali.



Gambar 1.8 Layar Inventory

Untuk mengetahui *flow* dari game ini, berikut ini adalah cara kerja permainan Diamonds.

1. Pertama, setiap pemain (bot) akan ditempatkan pada *board* secara *random*. Masing-masing bot akan mempunyai *home base,* serta memiliki *score* dan *inventory* awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil *diamond-diamond* yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, *diamond* yang berwarna merah memiliki 2 poin dan *diamond* yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah *inventory*, dimana *inventory* berfungsi sebagai tempat penyimpanan sementara *diamond* yang telah diambil. *Inventory* ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke *home base*.
5. Apabila bot menuju ke posisi *home base*, *score* bot akan bertambah senilai *diamond* yang tersimpan pada *inventory* dan *inventory* bot akan menjadi kosong kembali.

6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke *home base* dan semua *diamond* pada *inventory* bot B akan hilang, diambil masuk ke *inventory* bot A (istilahnya *tackle*).
7. Selain itu, terdapat beberapa fitur tambahan seperti *teleporter* dan *red button* yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. *Score* masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

Dalam tugas besar ini, mahasiswa diminta untuk bekerja dalam kelompok minimal dua orang dan maksimal tiga orang, dengan lintas kelas dan lintas kampus diperbolehkan. Tujuan dari tugas ini adalah untuk mengimplementasikan algoritma Greedy pada bot permainan Diamonds dengan strategi yang dirancang untuk memenangkan permainan. Setiap kelompok diminta untuk mengembangkan strategi Greedy terbaik yang berkaitan dengan fungsi objektif permainan, yaitu mengumpulkan diamond sebanyak mungkin dan mencegah diamond tersebut diambil oleh bot lain.

Strategi Greedy yang diterapkan oleh setiap kelompok harus dijelaskan secara eksplisit dalam laporan, dan kode program yang sesuai dengan strategi tersebut harus disertakan. Mahasiswa dilarang menggunakan kode program yang diunduh dari internet; setiap kelompok diharapkan untuk membuat program mereka sendiri dengan menggunakan kreativitas mereka. Program harus dapat dijalankan pada game engine yang ditentukan dan dapat bersaing dengan bot dari kelompok lain.

Setiap implementasi strategi Greedy harus dijelaskan dengan komentar yang jelas dalam kode program. Selain itu, terdapat bonus poin untuk kelompok yang membuat video tentang aplikasi Greedy pada bot dan simulasinya dalam permainan, dengan menampilkan wajah dari setiap anggota kelompok. Asistensi tugas besar bersifat opsional, namun jika diperlukan, mahasiswa dapat meminta bimbingan dari asisten yang dipilih.

Kelompok yang telah membuat bot akan berkompetisi dengan kelompok lain dalam kompetisi yang disaksikan oleh seluruh peserta kuliah. Hadiah menarik akan diberikan kepada kelompok yang memenangkan kompetisi. Informasi terkait pendataan kelompok, asistensi, demo, dan pengumpulan laporan telah disediakan melalui tautan yang telah disediakan.

Seluruh laporan tugas besar harus dikumpulkan dalam format PDF melalui tautan yang telah disediakan paling lambat pada tanggal 1 Juni 2025. Isi dari repository yang digunakan untuk menyimpan program harus mengikuti struktur yang telah ditentukan, termasuk folder src untuk source code, folder doc untuk laporan, dan README yang berisi penjelasan singkat algoritma Greedy yang diimplementasikan, persyaratan program, langkah-langkah untuk penggunaan, dan informasi tentang pembuat program.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy adalah salah satu teknik dalam pemrograman algoritmik yang digunakan untuk menyelesaikan masalah optimasi. Algoritma ini bekerja dengan cara memilih solusi lokal terbaik pada setiap langkah dengan harapan bahwa keputusan tersebut akan menghasilkan solusi global yang optimal.

Ciri utama dari algoritma greedy adalah bahwa keputusan yang diambil pada satu langkah tidak akan diubah pada langkah-langkah berikutnya. Dengan kata lain, algoritma ini tidak melakukan penelusuran balik (backtracking) atau pertimbangan ulang atas pilihan sebelumnya [1].

Meskipun algoritma greedy tidak selalu memberikan solusi optimal untuk semua jenis permasalahan, namun pada beberapa kasus tertentu seperti *Huffman coding*, *Prim's algorithm*, dan *Dijkstra's algorithm*, pendekatan ini terbukti efisien dan memberikan hasil yang optimal [2]. Dalam implementasinya, algoritma greedy biasanya terdiri dari tiga komponen utama :

1. Kandidat solusi, yaitu himpunan semua pilihan yang memungkinkan.
2. Fungsi seleksi, untuk memilih elemen terbaik berikutnya.
3. Fungsi kelayakan, untuk memutuskan apakah elemen terpilih dapat digunakan sebagai bagian dari solusi akhir.

2.2 Cara Kerja Program

Program bekerja dengan mengimplementasikan algoritma greedy berbobot (multi-weighted greedy) untuk menentukan aksi bot dalam permainan Diamonds. Setiap langkah, bot mengevaluasi posisi berlian, musuh, teleporter, dan tombol merah untuk memilih target terbaik yang dapat dicapai secara efisien dan aman. Bot menggunakan berbagai bobot seperti keamanan, kepadatan berlian (cluster), efisiensi jalur, dan tekanan kompetitif untuk menghitung skor dari setiap kemungkinan target. Setelah target dipilih, bot akan bergerak selangkah demi selangkah menuju lokasi tersebut.

1. **Cara Implementasi Program**

Program ini diimplementasikan sebagai kelas Python bernama MultiWeightedGreedyLogic, yang merupakan turunan dari BaseLogic. Beberapa fitur penting :

- Perhitungan skor DHG (Dynamic Heuristic Greedy) dilakukan di calculate_dhg_score, yang menggabungkan berbagai faktor.
- Evaluasi posisi target dilakukan di find_optimal_target.
- Fungsi next_move menentukan arah gerakan berdasarkan target optimal.
- Fitur tambahan seperti anti-stuck, endgame awareness, dan competitive pressure memperkaya keputusan bot.

Bot berjalan secara deterministik berdasarkan evaluasi greedy terhadap semua berlian yang tersedia, lalu bergerak ke target dengan skor tertinggi.

2. **Menjalankan Bot Program**

Bot dijalankan dalam framework permainan yang sudah disediakan, dengan cara :

- Menambahkan file MultiWeightedGreedyLogic ke folder game/logic.
- Mengubah konfigurasi bot untuk menggunakan kelas MultiWeightedGreedyLogic.
- Menjalankan game engine menggunakan perintah seperti : python3 main.py
- Bot akan otomatis bergerak berdasarkan strategi greedy berbobot setiap giliran.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Proses mapping dilakukan dengan mengubah elemen-elemen permainan menjadi informasi yang bisa dievaluasi algoritma greedy :

- Berlian dimapping sebagai target yang memiliki nilai (points) dan posisi (Position).
- Musuh dimapping sebagai ancaman berdasarkan jarak dan jumlah berlian yang mereka bawa.
- Teleportasi dimapping sebagai jalur alternatif untuk efisiensi.
- Waktu tersisa dan jumlah berlian yang dibawa dimapping untuk strategi endgame atau keputusan pulang ke base.
- Semua data ini kemudian digunakan untuk menghitung skor pada tiap posisi target menggunakan fungsi calculate_dhg_score.

3.2 Eksplorasi Alternatif Solusi Greedy

Beberapa alternatif strategi greedy yang kami pertimbangkan ialah :

1. Greedy klasik : Selalu memilih berlian terdekat tanpa mempertimbangkan nilai atau risiko.
2. Greedy nilai tertinggi : Memilih berlian dengan poin terbanyak, tanpa mempertimbangkan jarak.
3. Greedy berbobot (yang dipakai) : Menggabungkan banyak faktor seperti nilai, jarak, cluster, keamanan, dan tekanan kompetisi.
4. Greedy berbasis prediksi musuh : Menambahkan evaluasi gerakan musuh untuk menghindari atau menyerang.
5. Greedy berbasis AHP (Analytical Hierarchy Process) : Teoretis, bobot ditentukan lewat perbandingan berpasangan.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Efisiensi :

- Strategi greedy berbobot hanya mengevaluasi kandidat target berdasarkan heuristik lokal → lebih cepat daripada algoritma global seperti A*(algoritma pencarian jalur yang lebih rumit karena menggunakan semua kemungkinan jalur dan menghitung langkah terbaik untuk mencapai tujuan dengan meminimalkan total biaya. Proses ini memerlukan lebih banyak waktu komputasi karena mempertimbangkan seluruh ruang solusi).
- Bot menghitung skor target tanpa melakukan penelusuran jalur penuh → hemat waktu komputasi.

Efektivitas :

- Dengan mempertimbangkan ancaman, waktu, dan kompetisi, strategi ini lebih adaptif terhadap dinamika permainan.
- Bot mampu menyeimbangkan antara pengambilan risiko dan keamanan, khususnya dalam fase akhir permainan.

Dibanding greedy sederhana, strategi ini jauh lebih efisien dalam mengambil keputusan yang relevan secara konteks.

3.4 Strategi Greedy yang Dipilih

Strategi greedy yang dipilih adalah Multi-Weighted Dynamic Heuristic Greedy (DHG). Strategi ini dipilih karena :

- Menggabungkan nilai berlian, jarak, keamanan, kepadatan, dan kondisi kompetitif dalam satu fungsi skor.
- Lebih stabil dan konsisten untuk berbagai kondisi permainan.
- Mampu menangani fase early-game, mid-game, hingga endgame dengan penyesuaian otomatis.

Dengan bobot-bobot yang disesuaikan, bot mampu mengambil keputusan optimal pada setiap langkah tanpa harus mengevaluasi semua jalur secara eksplisit.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

multi_weighted.py

```
## Inisialisasi
```
CLASS MultiWeightedGreedyLogic EXTENDS BaseLogic:
    INITIALIZE:
        goal_position = NULL
        static_goals = []
        static_goal_teleport = NULL
        path_cache = {}
        last_board_hash = ""
        previous_position = NULL
        stuck_counter = 0

        // Parameter optimasi
        SAFETY_WEIGHT = 2.5
        CLUSTER_WEIGHT = 2.0
        TIME_PRESSURE_THRESHOLD = 15.0
        DANGER_RADIUS = 4
        TACKLE_OPPORTUNITY_WEIGHT = 3.0
        MIN_DIAMONDS_FOR_RED = 3
        BASE_RETURN_BUFFER = 3
        DIAMOND_CLUSTER_RADIUS = 3
        ENDGAME_TIME_THRESHOLD = 30.0
        COMPETITION_AWARENESS_FACTOR = 1.5
```


## Fungsi Utama - next_move()
```
FUNCTION next_move(current_bot, board):
    // Reset goals jika sudah di base
    IF current_bot.position == current_bot.base:
        CLEAR static_goals
        SET static_goal_teleport = NULL
```
```

```
        CLEAR path_cache

    // Cari target optimal menggunakan enhanced DHG
    best_target, best_distance = find_optimal_target(current_bot,
board)

    // Tentukan tujuan berdasarkan kondisi
    IF best_target is NULL:
        SET goal_position = current_bot.base
    ELSE IF should_return_to_base_enhanced(current_bot, board,
best_distance):
        SET goal_position = current_bot.base
    ELSE:
        SET goal_position = best_target

    // Hitung arah gerakan
    dx, dy = calculate_direction(current_bot.position,
goal_position)

    // Validasi gerakan
    IF dx == 0 AND dy == 0:
        RETURN default_movement_based_on_position()

    // Cegah gerakan diagonal
    IF abs(dx) == abs(dy):
        RETURN dx, 0  // Prioritas horizontal

    RETURN dx, dy
```

Pencarian Target Optimal

```
FUNCTION find_optimal_target(current_bot, board):
    current_position = current_bot.position
    best_score = 0
    best_target = NULL
    best_distance = INFINITY

    // Mekanisme anti-stuck
    IF previous_position == current_position:
```

```
        INCREMENT stuck_counter
    ELSE:
        RESET stuck_counter = 0

    SET previous_position = current_position

    // Evaluasi setiap diamond
    FOR EACH diamond IN board.diamonds:
        // Cek batasan inventori
        IF inventory_constraint_violated(current_bot, diamond):
            CONTINUE

        // Cek kelayakan waktu-jarak
        distance = calculate_manhattan_distance(current_position,
diamond.position)
        time_left = current_bot.milliseconds_left / 1000
        required_time = distance * 2 + 2

        IF required_time > time_left:
            CONTINUE

        // Hitung DHG score
        dhg_score = calculate_dhg_score(current_position,
diamond.position,
                                        diamond.points, board,
current_bot)

        // Terapkan penalti stuck
        IF stuck_counter > 3:
            dhg_score *= (1 - stuck_counter * 0.1)

        // Update target terbaik
        IF dhg_score > best_score:
            best_score = dhg_score
            best_target = diamond.position
            best_distance = distance

    // Evaluasi teleporter
    FOR EACH teleporter IN board.teleporters:
        other_teleport = find_paired_teleporter(teleporter, board)
```

```
        IF other_teleport is NULL:
            CONTINUE

        teleport_distance =
calculate_manhattan_distance(current_position, teleporter.position)
        accessible_diamonds = []

        FOR EACH diamond IN board.diamonds:
            post_teleport_distance =
calculate_manhattan_distance(other_teleport.position,
diamond.position)
            total_distance = teleport_distance +
post_teleport_distance

            IF total_distance <= 10 AND NOT
inventory_constraint_violated(current_bot, diamond):
                ADD (diamond, total_distance) TO accessible_diamonds

        IF accessible_diamonds NOT EMPTY:
            SORT accessible_diamonds BY distance
            best_diamond = accessible_diamonds[0]

            dhg_score = calculate_dhg_score(other_teleport.position,
best_diamond.position,
                                            best_diamond.points,
board, current_bot) * 0.9

            IF dhg_score > best_score:
                best_score = dhg_score
                best_target = teleporter.position
                best_distance = total_distance
                static_goal_teleport = teleporter

    // Evaluasi red button
    red_button_score = evaluate_red_button_strategy(board,
current_bot)
    IF red_button_score > best_score:
        red_button = find_red_button(board)
        IF red_button EXISTS AND distance_to_red_button <= 8:
            best_target = red_button.position
```

```
            best_distance =
calculate_manhattan_distance(current_position, red_button.position)

    RETURN best_target, best_distance
```

Perhitungan DHG Score
```
FUNCTION calculate_dhg_score(current_pos, target_pos, target_value,
board, current_bot):
    manhattan_distance = calculate_manhattan_distance(current_pos,
target_pos)
    IF manhattan_distance == 0:
        RETURN INFINITY

    // Perhitungan komponen score
    cluster_weight = calculate_enhanced_cluster_weight(target_pos,
board, current_bot)
    safety_score = calculate_safety_score(target_pos, board,
current_bot)
    tackle_opportunity = calculate_tackle_opportunity(target_pos,
board, current_bot)
    path_efficiency = calculate_path_efficiency(current_pos,
target_pos, board)

    // Tekanan waktu
    time_left = current_bot.milliseconds_left / 1000
    IF is_endgame(current_bot):
        time_pressure = 2.0 + (ENDGAME_TIME_THRESHOLD - time_left) /
10.0
        safety_score *= 1.5
    ELSE IF time_left < TIME_PRESSURE_THRESHOLD:
        time_pressure = 1.0 + (TIME_PRESSURE_THRESHOLD - time_left)
/ TIME_PRESSURE_THRESHOLD
    ELSE:
        time_pressure = 1.0

    // Faktor inventori
    inventory_factor =
calculate_inventory_factor(current_bot.diamonds, target_value)
```

```
    // Faktor kompetitif
    competitive_pressure = calculate_competitive_pressure(board,
current_bot)

    // Perhitungan DHG Score final
    value_factor = (target_value * inventory_factor +
                    cluster_weight * CLUSTER_WEIGHT +
                    tackle_opportunity * TACKLE_OPPORTUNITY_WEIGHT) *
competitive_pressure

    distance_factor = manhattan_distance * time_pressure /
path_efficiency

    dhg_score = (value_factor * safety_score * SAFETY_WEIGHT) /
distance_factor

    RETURN dhg_score
```

Keputusan Kembali ke Base

```
FUNCTION should_return_to_base_enhanced(current_bot, board,
best_diamond_distance):
    base = current_bot.base
    base_distance =
calculate_manhattan_distance(current_bot.position, base)
    time_left = current_bot.milliseconds_left / 1000

    // Kondisi kritis
    IF current_bot.diamonds >= 5:
        RETURN TRUE

    // Manajemen waktu
    time_buffer = IF is_endgame(current_bot) THEN 5 ELSE 3
    IF base_distance >= time_left - time_buffer:
        RETURN TRUE

    // Strategi endgame
    IF is_endgame(current_bot) AND current_bot.diamonds >= 2:
```

```
        RETURN TRUE

    // Threshold inventori dinamis
    IF current_bot.diamonds >= 4:
        RETURN TRUE

    // Penilaian risiko
    enemies_nearby = 0
    enemy_threat_level = 0
    high_value_enemies = 0

    FOR EACH enemy_bot IN board.bots:
        IF enemy_bot.id == current_bot.id:
            CONTINUE

        distance =
calculate_manhattan_distance(current_bot.position,
enemy_bot.position)
        IF distance <= 4:
            INCREMENT enemies_nearby
            threat_weight = (5 - distance) * (1 + enemy_bot.diamonds
* 0.3)
            enemy_threat_level += threat_weight

            IF enemy_bot.diamonds >= 3:
                INCREMENT high_value_enemies

    // Kembali jika terancam
    IF enemy_threat_level > 6 AND current_bot.diamonds >= 2:
        RETURN TRUE

    IF high_value_enemies >= 2 AND current_bot.diamonds >= 1:
        RETURN TRUE

    // Opportunity cost kompetitif
    competitive_factor = calculate_competitive_pressure(board,
current_bot)
    distance_threshold = IF competitive_factor > 1.2 THEN 0.9 ELSE
0.8
```

```
    IF current_bot.diamonds >= 3 AND base_distance <
best_diamond_distance * distance_threshold:
        RETURN TRUE

    // Return berdasarkan kelangkaan
    diamond_scarcity_threshold = IF competitive_factor > 1.3 THEN 4
ELSE 3
    IF count(board.diamonds) <= diamond_scarcity_threshold AND
current_bot.diamonds >= 2:
        RETURN TRUE

    RETURN FALSE
```

Fungsi Pendukung
```
FUNCTION calculate_safety_score(position, board, current_bot):
    safety = 1.0
    total_threat = 0

    FOR EACH enemy_bot IN board.bots:
        IF enemy_bot.id == current_bot.id:
            CONTINUE

        distance = calculate_manhattan_distance(position,
enemy_bot.position)
        IF distance <= DANGER_RADIUS:
            enemy_to_base =
calculate_manhattan_distance(enemy_bot.position, enemy_bot.base)

            base_threat = 1.0 + (enemy_bot.diamonds * 0.2)

            IF enemy_bot.diamonds >= 3 AND enemy_to_base < distance:
                base_threat *= 0.4

            distance_factor = (DANGER_RADIUS - distance) /
DANGER_RADIUS
            threat_level = base_threat * distance_factor
            total_threat += threat_level
```

```
    safety = 1.0 / (1.0 + total_threat * 0.3)
    RETURN max(safety, 0.02)

FUNCTION calculate_enhanced_cluster_weight(position, board,
current_bot):
    weight = 0
    red_diamond_bonus = 0
    competitive_multiplier = calculate_competitive_pressure(board,
current_bot)

    red_clusters = count_red_diamonds_in_cluster(position, board)

    FOR EACH diamond IN board.diamonds:
        distance = calculate_manhattan_distance(position,
diamond.position)
        IF distance <= 5:
            decay_factor = exp(-distance / 1.5)
            base_value = diamond.points * decay_factor

            IF diamond.points == 2:
                red_diamond_bonus += base_value * (0.7 +
red_clusters * 0.1)

            weight += base_value

    total_weight = (weight + red_diamond_bonus) *
competitive_multiplier
    RETURN total_weight

FUNCTION calculate_competitive_pressure(board, current_bot):
    scores = [bot.score FOR bot IN board.bots]
    IF length(scores) <= 1:
        RETURN 1.0

    SORT scores DESCENDING
    leader_score = scores[0]
    our_score = current_bot.score

    IF leader_score == 0:
        RETURN 1.0
```

```
    score_ratio = our_score / leader_score
    RETURN max(0.5, 2.0 - score_ratio)

FUNCTION evaluate_red_button_strategy(board, current_bot):
    red_button = find_red_button(board)
    IF red_button is NULL:
        RETURN 0

    total_diamonds = count(board.diamonds)
    red_diamonds = count_red_diamonds(board.diamonds)
    time_left = current_bot.milliseconds_left / 1000

    competitive_pressure = calculate_competitive_pressure(board,
current_bot)
    current_ranking = get_score_ranking(board, current_bot)

    scarcity_factor = max(0, (12 - total_diamonds) / 12.0) *
competitive_pressure
    red_ratio = red_diamonds / max(total_diamonds, 1)
    optimal_red_ratio = 0.35
    red_ratio_factor = max(0, (optimal_red_ratio - red_ratio) /
optimal_red_ratio)

    total_bots = count(board.bots)
    ranking_pressure = (current_ranking - 1) / max(total_bots - 1,
1)

    time_factor = calculate_time_factor(time_left)

    button_distance =
calculate_manhattan_distance(red_button.position,
current_bot.position)
    distance_factor = max(0.3, 1.0 - button_distance / 10.0)
    red_button_score = (scarcity_factor * 0.25 +
                        red_ratio_factor * 0.25 +
                        ranking_pressure * 0.3 +
                        time_factor * 0.2) * 18 * distance_factor

    RETURN red_button_score
```

2. Penjelasan Alur Program

A. Inisialisasi

1. Bot dimulai dengan pengaturan parameter optimasi seperti bobot keamanan, deteksi cluster, dan threshold waktu
2. Sistem anti-stuck diinisialisasi untuk mencegah bot terjebak dalam loop

B. Fungsi Utama (next_move)

1. Reset Goals : Jika bot berada di base, semua tujuan sebelumnya direset
2. Pencarian Target : Mencari target optimal menggunakan algoritma DHG yang disempurnakan
3. Pengambilan Keputusan : Menentukan apakah harus kembali ke base atau menuju target
4. Perhitungan Gerakan : Menghitung arah gerakan yang optimal
5. Validasi : Memastikan gerakan valid (tidak diagonal, tidak diam di tempat)

C. Algoritma Pencarian Target (find_optimal_target)

1. Evaluasi Diamond :
    - Cek batasan inventori (tidak mengambil red diamond jika sudah penuh)
    - Validasi kelayakan waktu vs jarak
    - Hitung DHG score untuk setiap diamond
    - Terapkan mekanisme anti-stuck
2. Evaluasi Teleporter :
    - Cari teleporter berpasangan
    - Evaluasi diamond yang dapat diakses setelah teleport
    - Hitung efisiensi jalur teleporter
3. Evaluasi Red Button :
    - Analisis kondisi game untuk strategi red button
    - Pertimbangkan timing, posisi ranking, dan kelangkaan diamond

D. Perhitungan DHG Score

1. DHG Score mengintegrasikan berbagai faktor :
    - Nilai Target : Poin diamond + faktor inventori
    - Cluster Weight : Bobot berdasarkan kumpulan diamond di sekitar
    - Safety Score : Penilaian keamanan berdasarkan posisi musuh
    - Tackle Opportunity : Peluang untuk tackle musuh
    - Path Efficiency : Efisiensi jalur (termasuk teleporter)
    - Time Pressure : Tekanan waktu remaining
    - Competitive Pressure : Tekanan kompetitif berdasarkan skor

E. Sistem Keamanan dan Anti-Stuck

1. Safety Score
    - Menghitung ancaman dari semua musuh dalam radius bahaya
    - Mempertimbangkan inventori musuh dan jarak ke base mereka
    - Memberikan skor keamanan yang lebih rendah untuk posisi berbahaya
2. Anti-Stuck Mechanism
    - Melacak posisi sebelumnya untuk mendeteksi bot yang stuck
    - Memberikan penalti pada target yang menyebabkan stuck
    - Menambah randomness untuk memecah cycle

F. Strategi Kembali ke Base

Bot akan kembali ke base jika :

- Inventori penuh (≥5 diamond)
- Waktu hampir habis dengan buffer keamanan
- Dalam fase endgame dengan minimal 2 diamond
- Terancam oleh banyak musuh
- Jarak ke base lebih efisien daripada ke diamond terdekat
- Diamond langka dan sudah memiliki cukup diamond

G. Adaptasi Kompetitif

1. Competitive Pressure
    - Menghitung tekanan kompetitif berdasarkan posisi skor
    - Meningkatkan agresivitas jika tertinggal
    - Mempengaruhi semua keputusan strategis
2. Endgame Strategy
    - Beralih ke strategi konservatif saat waktu < 30 detik
    - Prioritas keamanan dan efisiensi
    - Threshold return ke base lebih rendah

H. Optimasi Lanjutan

1. Red Button Strategy
    - Evaluasi timing optimal untuk menekan red button
    - Pertimbangkan rasio red diamond, scarcity, ranking, dan competitive pressure
    - Hanya aktif pada kondisi tertentu
2. Cluster Analysis
    - Deteksi kumpulan diamond untuk maksimalisasi efisiensi
    - Bonus khusus untuk red diamond cluster
    - Decay factor berdasarkan jarak
3. Tackle Opportunity
    - Mengevaluasi peluang untuk tackle musuh
    - Prediksi pergerakan musuh
    - Prioritas target berdasarkan inventori musuh

Bot ini dirancang untuk menjadi sangat kompetitif dengan kemampuan adaptasi yang tinggi terhadap berbagai situasi permainan yang dinamis.

4.2 Struktur Data yang Digunakan

Dalam program ini, beberapa struktur data digunakan untuk menunjang pengambilan keputusan bot, antara lain :

1. Board
   Mewakili kondisi permainan saat ini, termasuk informasi posisi berlian, bot lain, base, dan objek khusus seperti red button atau teleporter.
2. GameObject
   Objek umum yang digunakan untuk mewakili bot, diamond, base, dan lainnya. Memiliki atribut seperti position, properties, dan id.
3. Position
   Tipe data untuk menyimpan posisi objek dalam bentuk koordinat (x, y) di papan permainan.
4. Dictionary (dict)
   Digunakan untuk :
   - Menyimpan cache jalur (path_cache)
   - Menyimpan skor atau evaluasi dari target
   - Menyimpan pasangan teleporter dan objek red button
5. List (list)
   Digunakan untuk :
   - Menyimpan posisi target statis (static_goals)
   - Menyimpan daftar diamond dan bot untuk evaluasi
6. Variabel bantu dan scalar
   - stuck_counter: menghitung seberapa lama bot tidak berpindah posisi.
   - SAFETY_WEIGHT, CLUSTER_WEIGHT, dll: digunakan sebagai bobot dalam perhitungan skor greedy (DHG Score).

4.3 Pengujian Program

1. Skenario Pengujian

Beberapa skenario pengujian dilakukan untuk mengevaluasi performa bot :

1. Skenario 1 : Early Game, Banyak Diamond Biru
   Menguji apakah bot dapat memilih target yang mudah dijangkau dan efisien tanpa banyak gangguan musuh.
2. Skenario 2 : Mid Game, Red Diamond Muncul, Banyak Musuh Dekat
   Mengamati apakah bot lebih berhati-hati dan mampu menghindari musuh saat inventori penuh.

3. Skenario 3: End Game, Waktu Hampir Habis, Inventori Hampir Penuh
   Menguji apakah bot memilih pulang ke base lebih awal untuk mengamankan poin.
4. Skenario 4: Keputusan Tackle vs Aman
   Menguji apakah bot memilih untuk menyerang bot lain yang membawa diamond jika peluang tackle menguntungkan.
5. Skenario 5: Evaluasi Red Button Strategy
   Menilai kapan bot memilih menuju red button ketika kondisi permainan mendukung regenerasi diamond.

2. Hasil Pengujian dan Analisis

Tabel 4.3 Hasil Pengujian dan Analisis

Skenario	Perilaku Bot	Hasil	Analisis
1	Memilih diamond terdekat, cepat kembali ke base	Berhasil	Strategi greedy berfungsi baik di awal
2	Menghindari musuh, memilih jalur aman	Berhasil	Safety score berjalan efektif
3	Kembali ke base tepat waktu	Berhasil	Endgame threshold optimal
4	Tackle bot musuh ketika menguntungkan	Berhasil	Tackle opportunity dimanfaatkan
5	Menuju red button saat rasio red rendah	Berhasil	Red button strategy responsif

Secara umum, bot menunjukkan adaptasi tinggi terhadap perubahan kondisi permainan, terutama karena adanya bobot dinamis dan strategi endgame.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada laporan tugas besar ini, kami berhasil mengimplementasikan strategi Multi-Weighted Dynamic Heuristic Greedy (DHG) dalam pembuatan bot permainan *Diamonds*. Algoritma ini mempertimbangkan banyak faktor seperti nilai diamond, efisiensi jalur, ancaman musuh, tekanan waktu, serta tekanan kompetitif. Bot kami mampu :

- Mengambil keputusan optimal secara real-time.
- Menyesuaikan strategi sesuai dengan fase permainan (early–mid–endgame).
- Menghindari situasi berbahaya dan memaksimalkan peluang tackle.
- Menggunakan fitur red button dan teleporter secara cerdas.

Strategi greedy berbobot ini terbukti efektif dan efisien dalam menghadapi tantangan permainan yang dinamis.

5.2 Saran

Beberapa saran pengembangan ke depan :

1. Integrasi Machine Learning (ML)
   Agar bot bisa belajar dari data permainan sebelumnya untuk meningkatkan keputusan secara adaptif.
2. Perbaikan Pathfinding
   Meskipun strategi greedy cepat, menambahkan pathfinding (seperti A* atau Dijkstra untuk beberapa kasus) bisa membantu saat kondisi sangat kompleks.
3. Adaptasi Real-Time
   Menambahkan sistem bobot dinamis berbasis statistik real-time, seperti heatmap posisi musuh atau diamond.
4. Penggunaan Visualisasi Debugging
   Agar pengujian dan analisis strategi lebih mudah dipahami.

LAMPIRAN

A. **Repository Github (link)**

   https://github.com/MarioSitepu/Tubes1_Sigma

B. **Video Penjelasan (link GDrive)**

   https://drive.google.com/drive/folders/1qKpiqNniBMVWyHQE381bpgWGpIluPSYZ **(Drive)**

   https://youtu.be/5xt24WoTiqY?si=Dn72rNd5wA7xQ_uZ **(Youtube)**

DAFTAR PUSTAKA

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

[2] E. Horowitz, S. Sahni, and S. Rajasekaran, *Fundamentals of Computer Algorithms*, 2nd ed., Hyderabad: Universities Press, 2008.

[3] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed., Pearson, 2010.

[4] J. Kleinberg and É. Tardos, *Algorithm Design*, 1st ed., Boston: Pearson Education, 2006.