IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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

Oleh: Kelompok Stigma Algoritma

Dhian Adi Nugraha	121140055
Ryanda Aditya Irawan	123140144
Muhammad Arkan Saktiawan	123140166

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..... 3
BAB II
LANDASAN TEORI..... 4
2.1 Dasar Teori..... 4
1. Cara Implementasi Program..... 5
2. Menjalankan Bot Program..... 5
BAB III
APLIKASI STRATEGI GREEDY..... 7
3.1 Proses Mapping..... 7
3.2 Eksplorasi Alternatif Solusi Greedy..... 7
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy..... 9
3.4 Strategi Greedy yang Dipilih..... 9
BAB IV
IMPLEMENTASI DAN PENGUJIAN..... 12
4.1 Implementasi Algoritma Greedy..... 12
1. Pseudocode..... 12
2. Penjelasan Alur Program..... 17
4.2 Struktur Data yang Digunakan..... 19
4.3 Pengujian Program..... 19
1. Skenario Pengujian..... 19
2. Hasil Pengujian dan Analisis..... 20
BAB V
KESIMPULAN DAN SARAN..... 22
5.1 Kesimpulan..... 22
5.2 Saran..... 22
LAMPIRAN..... 23
DAFTAR PUSTAKA..... 24

BAB I
DESKRIPSI TUGAS

Diamonds merupakan suatu programming challenge yang mempertandingkan bot yang anda buat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy adalah pendekatan dalam pemrograman yang memecahkan persoalan optimasi dengan cara yang tampaknya rakus. Pendekatan ini berfokus pada pengambilan keputusan sekarang dengan harapan bahwa setiap langkah akan membawa kita lebih dekat ke solusi akhir yang optimal.

Dalam konteks greedy, kita selalu memilih opsi yang paling menguntungkan saat ini tanpa mempertimbangkan konsekuensi di masa depan. Ini mirip dengan mengambil sejumlah uang tunai yang tersedia dari mesin ATM tanpa memikirkan bagaimana pengeluaran itu akan memengaruhi saldo akhir .

Kegunaan utama dari algoritma greedy adalah untuk menemukan solusi optimal dalam persoalan optimasi dengan cepat. Pendekatan ini sangat berguna dalam banyak kasus di mana kita perlu memaksimalkan atau meminimalkan sesuatu dengan cara yang efisien. Contoh penerapannya termasuk perencanaan jadwal, pengkodean data, manajemen sumber daya, dan banyak lagi.

Terdapat beberapa jenis algoritma greedy yang digunakan dalam berbagai konteks, berikut adalah contohnya:

1. Huffman Coding

   Digunakan dalam kompresi data. Ini adalah metode yang efisien untuk mengurangi ukuran data dengan assign kode biner yang lebih pendek untuk karakter yang lebih sering muncul dalam teks.

2. Kruskal Algorithm

   Digunakan dalam masalah pohon minimal (Minimum Spanning Tree) pada grafik. Ini membantu menemukan subset dari semua edge dalam grafik yang membentuk pohon tanpa siklus dengan total bobot yang minimal.

3. Prim’s Algorithm

   Seperti Kruskal, Prim’s Algorithm juga digunakan dalam masalah pohon minimal, tetapi fokus pada membangun pohon dari satu simpul awal dengan memilih edge terkecil yang terhubung.

2.2 Cara Kerja Program

Bagaimana cara kerja program secara umum (bagaimana bot melakukan aksinya, bagaimana mengimplementasikan algoritma greedy ke dalam bot, bagaimana menjalankan bot, dll). Bagian ini boleh diisi bagian deskripsi umum cara kerjanya saja kemudian diperinci per sub-bagian lagi:

1. Cara Implementasi Program

Dalam pengambilan keputusan, bot dirancang untuk bergerak menuju diamond dengan strategi yang mempertimbangkan jarak dan nilai diamond secara dinamis. Pertama-tama, bot akan mencari diamond yang paling dekat dari posisinya saat ini. Jika terdapat beberapa diamond dengan jarak yang sama, maka bot akan memilih diamond dengan nilai tertinggi (prioritas berdasarkan nilai). Apabila ditemukan diamond dengan nilai 4, maka diamond tersebut akan langsung menjadi prioritas utama untuk diambil. Sebaliknya, jika tidak ditemukan diamond dengan nilai tinggi, maka prioritas akan disesuaikan dengan mengambil diamond dengan nilai terkecil yang tersedia. Ketika jumlah diamond yang telah dikumpulkan mencapai 5, maka bot akan langsung pulang menuju portal. Selain itu, jika jarak bot ke diamond lebih jauh dibandingkan jarak diamond tersebut ke portal, dan diamond tersebut juga lebih dekat ke portal, maka bot akan memilih langsung menuju portal untuk menyelesaikan misi secara efisien. Strategi ini memastikan bahwa bot tidak hanya mengejar nilai maksimal, tetapi juga memperhitungkan efisiensi waktu dan jarak.

2. Menjalankan Bot Program

Untuk menjalankan bot, Anda dapat memilih untuk menjalankan satu bot saja atau beberapa bot sekaligus, tergantung kebutuhan. Berikut adalah langkah-langkahnya:

- Menjalankan Satu Bot (Costumable)

  Jalankan perintah berikut di terminal untuk menjalankan satu bot dengan logic Greedy:

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

- Menjalankan Beberapa Bot Sekaligus

  Jika ingin menjalankan beberapa bot sekaligus, dapat menggunakan script:

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

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Proses mapping diawali dengan mendefinisikan tugas bot, yaitu bergerak di grid 2D untuk mengumpulkan diamond sebanyak mungkin dan kembali ke base saat kapasitas maksimal (5 diamond) tercapai. Setiap giliran, bot hanya boleh melangkah satu kali secara horizontal, vertikal, atau acak (untuk menghindari jebakan). Objek teleport dapat mempercepat pergerakan jika digunakan dengan tepat.

Masalah ini dimodelkan sebagai graf berbobot, dengan posisi grid sebagai vertex dan langkah antar posisi sebagai edge berbobot 1. Jika menggunakan teleport, bobot dihitung sebagai jumlah jarak Manhattan dari posisi bot ke pintu masuk teleport dan dari pintu keluar ke target. Heuristic yang digunakan adalah jarak Manhattan untuk memperkirakan kedekatan bot ke diamond atau teleport.

Keputusan bot bersifat lokal setiap giliran dan mempertimbangkan atribut seperti jumlah diamond yang dibawa, kapasitas maksimum, posisi base, serta lokasi diamond dan teleport. Bot akan: (1) kembali ke base jika kapasitas penuh; (2) menuju diamond terdekat jika masih bisa menampung; (3) menggunakan teleport jika lebih cepat; (4) memilih langkah valid menuju target atau bergerak acak jika terhalang.

Dengan demikian, setiap giliran, bot memetakan kondisi saat ini yang mencakup posisi, jumlah diamond yang sudah dikumpulkan, dan informasi objek di peta ke dalam satu target (goal_position), lalu mengambil satu langkah menuju target tersebut. Proses ini bersifat lokal sepenuhnya: bot tidak merencanakan seluruh jalur dari awal hingga seluruh diamond habis diambil, melainkan membuat keputusan secara adaptif di setiap giliran berdasarkan kondisi terkini dan kapasitas yang tersisa.

3.2 Eksplorasi Alternatif Solusi Greedy

Selain pendekatan “nearest-neighbor + teleport heuristic” yang digunakan pada kode, terdapat beberapa alternatif algoritma greedy yang dapat dieksplorasi untuk meningkatkan

performa bot. Setiap pendekatan membawa keunggulan dan kelemahan tersendiri tergantung konteks permainan, kondisi lingkungan, serta batasan seperti kapasitas maksimal bot.

1. Prioritas Diamond Berdasarkan Nilai (Value-aware Greedy)

   Pendekatan ini mempertimbangkan tidak hanya jarak diamond dari bot, tetapi juga nilai diamond tersebut (d.properties.value). Sebagai contoh, daripada memilih diamond terdekat tanpa mempertimbangkan nilainya, bot bisa menghitung rasio antara nilai dan jarak menggunakan rumus:

   $$\text{score}(d) = \text{value}(d) / \text{distance}(\text{bot}, d)$$

   Dengan strategi ini, bot akan lebih bijak dalam memilih target. Misalnya, jika ada diamond bernilai 3 pada jarak 4 dan diamond bernilai 1 pada jarak 1, maka diamond bernilai 3 mungkin akan dipilih karena rasio nilainya lebih tinggi. Keunggulan dari pendekatan ini adalah meningkatnya potensi total nilai yang dikumpulkan, namun kelemahannya terletak pada tambahan komputasi dan kemungkinan bot mengabaikan diamond yang dekat namun bernilai kecil.

2. Greedy Berdasarkan “Sisa Ruang Kapasitas” (Capacity-aware Greedy)

   Strategi ini mengatur pemilihan target berdasarkan sisa ruang penyimpanan bot. Misalnya, jika bot sudah membawa 4 diamond, maka ia hanya akan mempertimbangkan diamond dengan nilai 1 untuk menghindari kelebihan muatan.

   Strategi ini cukup efisien dan aman, terutama saat mendekati kapasitas maksimum. Namun, dalam kondisi tertentu, keputusan mengejar diamond bernilai lebih tinggi tetap bisa dipertimbangkan jika memiliki perhitungan risiko yang memadai.

3. Greedy “Panjang Rute Sisa” (Remaining-distance Greedy)

   Di sini, bot tidak hanya menghitung jarak ke target diamond, tetapi juga memperkirakan jarak dari posisi diamond ke base untuk mengoptimalkan efisiensi pulang. Heuristik yang digunakan:

   $$\text{cost}(d) = \text{distance}(\text{bot}, d) + \text{distance}(d, \text{base})$$

Dengan mempertimbangkan rute pulang, bot lebih jarang berada dalam kondisi tersesat jauh dari base setelah mencapai kapasitas penuh. Namun, seperti strategi sebelumnya, perhitungan tambahan untuk setiap diamond juga meningkatkan overhead komputasi.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Algoritma greedy menawarkan pendekatan yang cepat dan sederhana dalam pengambilan keputusan untuk permainan mengumpulkan diamond ini. Dalam permainan ini, strategi greedy seperti nearest-neighbor, value-aware, capacity-aware, dan remaining-distance greedy memiliki keunggulan dalam kecepatan eksekusi karena tidak memerlukan eksplorasi jalur secara menyeluruh. Strategi nearest-neighbor sangat efisien secara komputasi namun cenderung miopik(sifat melihat jangka pendek), sering kali mengabaikan solusi yang lebih menguntungkan. Sementara itu, value-aware dapat meningkatkan akumulasi nilai dengan memperhitungkan rasio antara nilai diamond dan jaraknya. Strategi capacity-aware menjaga efisiensi ruang dan menghindari kesalahan ambil diamond saat kapasitas hampir penuh, cocok untuk mengurangi resiko kelebihan muatan yang menyebabkan error. Sedangkan remaining-distance greedy lebih holistik(pendekatan dengan pertimbangan keseluruhan sistem) karena memperhitungkan total rute pulang ke base, sehingga cocok untuk skenario dimana efisiensi waktu pengumpulan menjadi krusial. Secara keseluruhan, pendekatan greedy memberikan trade-off yang baik antara kesederhanaan dan efektivitas, namun rentan terhadap solusi optimal di lingkungan yang kompleks tanpa penyesuaian heuristik(pendekatan yang praktis) yang cermat.

3.4 Strategi Greedy yang Dipilih

Melalui strategi greedy yang diimplementasikan dalam kode mengadopsi pendekatan nearest-neighbor dengan optimisasi tambahan melalui pengecekan teleport. Strategi ini dapat dijabarkan sebagai berikut:

1. Heuristik (Mapping State ke Value Function)

   Bot mendefinisikan state sebagai kombinasi dari posisi bot, jumlah berlian yang dibawa (props.diamonds), daftar berlian di papan, daftar teleport, dan posisi base. Fungsi heuristik kemudian menentukan possible_diamonds—yaitu semua berlian yang jika

diambil tidak akan membuat total berlian melebihi max_capacity. Secara khusus, jika props.diamonds == 4, hanya berlian dengan value = 1 yang dipertimbangkan.

- Jika tidak ada possible_diamonds, maka bot mengatur base sebagai tujuan.
- Jika ada, bot mencari berlian terdekat berdasarkan jarak Manhattan. Selanjutnya, ia mengevaluasi semua pasangan teleport dengan menghitung total jarak: jarak(bot → entry) + jarak(exit → berlian). Jika hasilnya lebih pendek dari jarak langsung ke berlian, maka bot memilih entry teleport sebagai goal_position.

2. Eksekusi Single-step Move

   Setelah goal_position ditentukan, bot memanggil fungsi get_direction(current, goal) untuk menghasilkan langkah (dx, dy) satu unit menuju tujuan. Jika langkah tersebut valid (board.is_valid_move), maka bot langsung melangkah. Jika tidak valid, ia mencoba hingga empat arah alternatif. Bila semua alternatif gagal, langkah dipilih secara acak dari arah yang tersedia.

   Bila bot sudah berada di goal_position (misalnya di pintu teleport), bot tetap melangkah ke sembarang arah valid agar tidak diam, karena teleport hanya aktif saat bot masuk ke sel teleport melalui langkah.

3. Game Flow-Cycle

   - **Mengambil Berlian:** Selama jumlah berlian yang dibawa belum mencapai kapasitas maksimum, bot terus mencari dan menuju berlian terdekat atau entry teleport terdekat.
   - **Kembali ke Base:** Jika kapasitas penuh atau tidak ada berlian yang bisa diambil, tujuan beralih ke props.base. Bot berjalan menuju base, dan setelah tiba, diasumsikan berlian otomatis dikumpulkan, dan props.diamonds di-reset ke nol.
   - **Loop:** Setelah berlian diserahkan, goal_position di-reset dan proses dimulai ulang dari langkah pertama.

Strategi ini dipilih karena beberapa alasan utama. Pertama, kesederhanaan implementasi: cukup dengan menambahkan beberapa fungsi pembantu seperti find_closest_diamond, group_teleports, dan get_direction, tanpa perlu menggunakan algoritma pathfinding kompleks seperti A*. Kedua, kinerja yang memadai: untuk papan dengan ukuran umum seperti 50×50 atau lebih kecil, pendekatan greedy ini sudah cukup efektif untuk meraih skor tinggi tanpa beban komputasi yang berat. Ketiga, kemampuan adaptasi yang baik: bot dapat menyesuaikan target secara dinamis jika kondisi papan berubah, seperti munculnya berlian baru atau ketika berlian diambil oleh bot lain. Keempat, strategi ini mampu memanfaatkan teleport secara efektif sebagai cara tambahan untuk mengurangi jarak tempuh, tanpa perlu struktur data rumit atau graf eksplisit.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
# Dhian Adi Nugraha (121140055)
# Ryanda Aditya Irawan (123140144)
# Muhammad Arkan Saktiawan (123140166)

import random
from typing import Optional, List, Dict

from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
from ..util import clamp


class MyBot(BaseLogic):
    def __init__(self):
        self.goal_position: Optional[Position] = None
        self.max_capacity = 5

    def get_direction(self, current_x, current_y, dest_x,
dest_y):
        if current_x == dest_x and current_y == dest_y:
            return random.choice([(1, 0), (-1, 0), (0, 1), (0,
-1)])
        delta_x = clamp(dest_x - current_x, -1, 1)
        delta_y = clamp(dest_y - current_y, -1, 1)
```

```
        if abs(delta_x) == abs(delta_y):
            if abs(dest_x - current_x) > abs(dest_y -
current_y):
                delta_y = 0
            else:
                delta_x = 0
        return (delta_x, delta_y)

     def group_teleports(self, teleports: List[GameObject]) ->
Dict[str, List[Position]]:
        pairs = {}
        for tp in teleports:
            pid = tp.properties.pair_id if tp.properties else
None
            if pid:
                if pid not in pairs:
                    pairs[pid] = []
                pairs[pid].append(tp.position)
        return pairs

    def find_closest_diamond(self, bot_pos: Position, diamonds:
List[GameObject]) -> Position:
        return min(
            diamonds,
            key=lambda d: abs(d.position.x - bot_pos.x) +
abs(d.position.y - bot_pos.y)
        ).position

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

```
        props = board_bot.properties
        current_position = board_bot.position
        diamonds = board.diamonds

        if props.diamonds >= self.max_capacity:
            if props.base and (props.base.x !=
current_position.x or props.base.y != current_position.y):
                self.goal_position = props.base
            else:
                for alt_dx, alt_dy in [(1, 0), (-1, 0), (0, 1),
(0, -1)]:
                    if board.is_valid_move(current_position,
alt_dx, alt_dy):
                        return alt_dx, alt_dy
                return random.choice([(1, 0), (-1, 0), (0, 1),
(0, -1)])

        if not diamonds:
            for dx, dy in [(1,0), (-1,0), (0,1), (0,-1)]:
                if board.is_valid_move(current_position, dx,
dy):
                    return dx, dy
            return random.choice([(1, 0), (-1, 0), (0, 1), (0,
-1)])

        if props.diamonds == 4:
            possible_diamonds = [
                d for d in diamonds
                if d.properties
                and getattr(d.properties, "value", 1) == 1
```

```
                and (props.diamonds + getattr(d.properties,
"value", 1)) <= self.max_capacity
            ]
        else:
            possible_diamonds = [
                d for d in diamonds
                if d.properties
                and (props.diamonds + getattr(d.properties,
"value", 1)) <= self.max_capacity
            ]

        if not possible_diamonds:
            if props.base and (props.base.x !=
current_position.x or props.base.y != current_position.y):
                self.goal_position = props.base
            else:
                for alt_dx, alt_dy in [(1, 0), (-1, 0), (0, 1),
(0, -1)]:
                    if board.is_valid_move(current_position,
alt_dx, alt_dy):
                        return alt_dx, alt_dy
                return random.choice([(1, 0), (-1, 0), (0, 1),
(0, -1)])
        else:
            nearest_diamond_pos =
self.find_closest_diamond(current_position, possible_diamonds)

            teleports = [g for g in board.game_objects if
g.type == "TeleportGameObject"]
            tp_pairs = self.group_teleports(teleports)
```

```
            direct_distance = abs(nearest_diamond_pos.x -
current_position.x) + abs(nearest_diamond_pos.y -
current_position.y)
            best_path_len = direct_distance
            best_tp_entry = None

            for pair in tp_pairs.values():
                if len(pair) != 2:
                    continue
                a, b = pair
                for entry, exit in [(a, b), (b, a)]:
                    dist_to_entry = abs(current_position.x -
entry.x) + abs(current_position.y - entry.y)
                    dist_from_exit = abs(exit.x -
nearest_diamond_pos.x) + abs(exit.y - nearest_diamond_pos.y)
                    total_dist = dist_to_entry + dist_from_exit
                    if total_dist < best_path_len:
                        best_path_len = total_dist
                        best_tp_entry = entry

            self.goal_position = best_tp_entry if best_tp_entry
else nearest_diamond_pos

        if current_position.x == self.goal_position.x and
current_position.y == self.goal_position.y:
            for dx, dy in [(1,0), (-1,0), (0,1), (0,-1)]:
                if board.is_valid_move(current_position, dx,
dy):
                    return dx, dy
```

```
            return random.choice([(1, 0), (-1, 0), (0, 1), (0,
-1)])

        dx, dy = self.get_direction(
            current_position.x,
            current_position.y,
            self.goal_position.x,
            self.goal_position.y,
        )

        if board.is_valid_move(current_position, dx, dy):
            return dx, dy
        else:
            for alt_dx, alt_dy in [(1, 0), (-1, 0), (0, 1), (0,
-1)]:
                if board.is_valid_move(current_position,
alt_dx, alt_dy):
                    return alt_dx, alt_dy
            return random.choice([(1, 0), (-1, 0), (0, 1), (0,
-1)])
```

2. **Penjelasan Alur Program**

- Inisiaisi Bot

  Bot yang dibuat, yaitu MyBot, merupakan turunan dari kelas BaseLogic. Bot ini memiliki dua properti utama yang disimpan: goal_position, yaitu posisi tujuan yang ingin dicapai bot saat ini, serta max_capacity, yang menunjukkan kapasitas maksimum diamond yang dapat dibawa oleh bot, yaitu sebanyak 5 buah.

- Fungsi Pendukung

- Terdapat beberapa fungsi pendukung untuk membantu logika pergerakan bot.
- Fungsi get_direction() digunakan untuk menentukan arah gerak menuju suatu koordinat tujuan (x, y) dengan langkah sejauh satu satuan. Apabila posisi saat ini sama dengan tujuan, maka bot akan bergerak secara acak.
- Fungsi group_teleports() mengelompokkan teleport berdasarkan pair_id sehingga bot dapat mencari pasangan teleport yang sesuai.
- Fungsi find_closest_diamond() digunakan untuk mencari diamond terdekat dari posisi bot dengan menggunakan metode Manhattan Distance.

- Fungsi Utama
  - Fungsi utama next_move() menentukan langkah yang harus diambil oleh bot setiap giliran.
  - Jika kapasitas diamond yang dibawa sudah penuh (≥ 5), maka bot akan diarahkan kembali ke base. Setelah tiba di base, bot akan bergerak secara acak di sekitarnya agar tetap aktif.
  - Jika tidak ada diamond yang tersisa di map, bot juga akan bergerak acak ke arah yang valid. Dalam kondisi membawa 4 diamond, bot hanya akan mengambil diamond kecil (dengan nilai 1) agar tidak melebihi kapasitas maksimum.
  - Jika masih terdapat diamond yang bisa diambil, bot akan memilih diamond terdekat sebagai target. Selanjutnya, bot akan memeriksa apakah terdapat jalur teleport yang dapat memperpendek jarak ke diamond tersebut. Jika jalur teleport lebih efisien, maka teleport akan digunakan, dan goal_position diatur ke posisi teleport atau ke diamond terdekat.
  - Dalam proses pergerakan menuju goal_position, jika bot sudah berada di posisi tujuan, maka ia akan bergerak acak untuk tetap aktif. Jika belum mencapai tujuan, bot akan melangkah ke arah goal_position dengan

bantuan fungsi get_direction(). Namun, apabila arah tersebut tidak valid, maka bot akan memilih arah lain yang valid secara acak.

4.2 Struktur Data yang Digunakan

Struktur Data	Digunakan Untuk
Position	Menyimpan koordinat objek di papan
GameObject	Mewakili entitas (bot, diamond, teleport)
Board	Informasi papan permainan & validasi gerakan
List	Kumpulan objek (diamond, teleport, dsb.)
Dict[str, List[Position]]	Kelompok teleport berdasarkan pasangan ID
Optional[Position]	Menyimpan tujuan bot (bisa None)

4.3 Pengujian Program

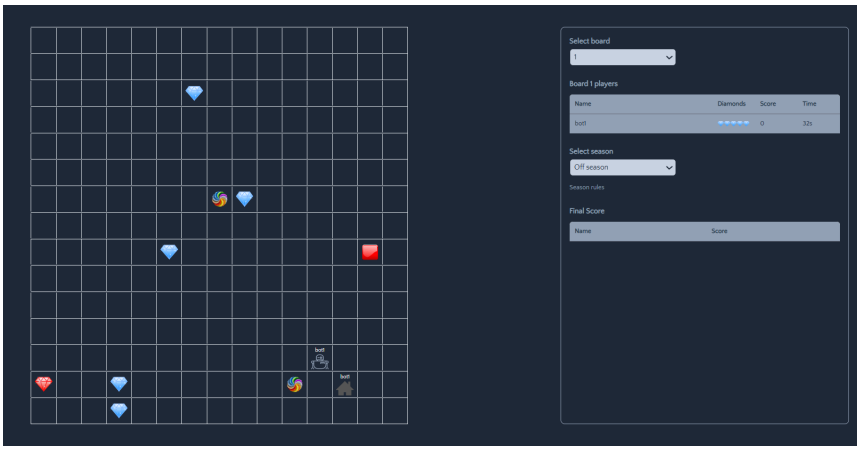
1. **Skenario Pengujian**

Pengujian dilakukan dengan menjalankan beberapa sesi permainan untuk melihat bagaimana bot merespons penyebaran diamond, baik dari segi jarak, value diamond, maupun penggunaan teleport. Dalam beberapa sesi bot dihadapkan dengan penyebaran diamond yang merata. Disesi lain, diamond bisa jadi tersebar tidak merata di satu sisi atau bahkan dekat dengan base. Dalam kondisi ini, pengamatan dilakukan untuk melihat apakah bot benar-benar selalu mengejar diamond terdekat dengan value terbesar, apakah bot akan kembali ke base ketika kapasitas penuh, serta apakah bot memanfaatkan fitur teleport yang tersedia.

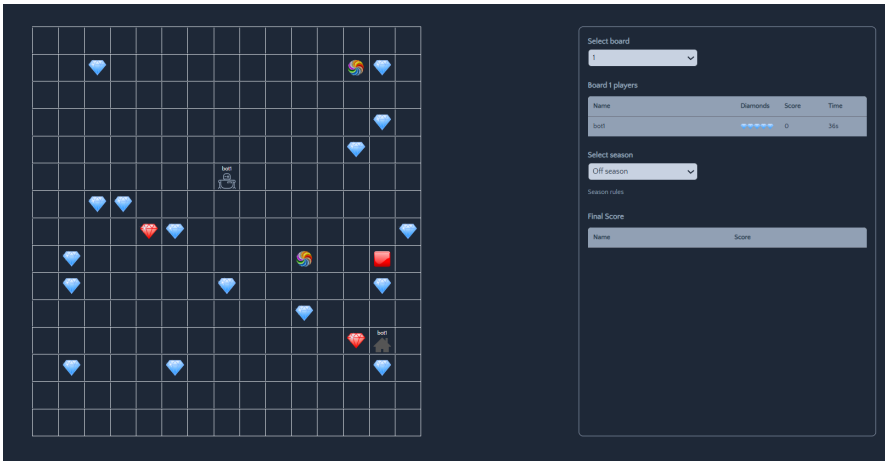
Karena ketidakpastian posisi diamond, pengujian juga melihat adaptasi bot terhadap perubahan lingkungan(board). Apakah bot tetap aktif mencari diamond yang memungkinkan diambil, atau menjadi pasif ketika kondisi tidak optimal. Dengan melakukan beberapa sesi pengujian, skenario ini dapat memberikan gambaran umum terkait konsistensi peforma dan

efektivitas strategi greedy yang digunakan bot dalam menghadapi kondisi acak dan dinamis dari game.

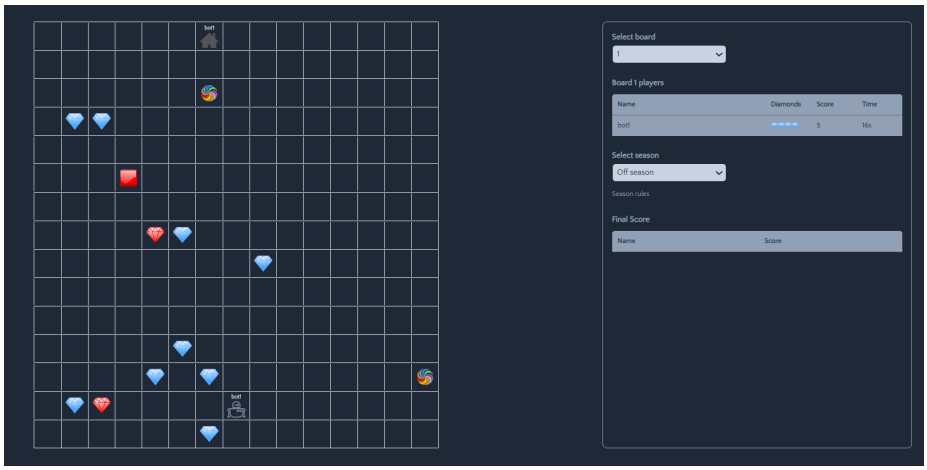
2. Hasil Pengujian dan Analisis



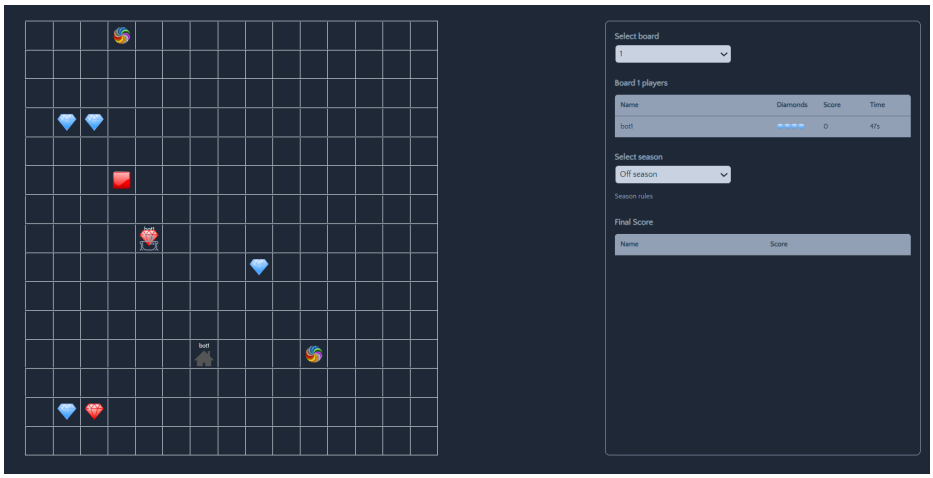
Di pengujian pertama ini bot berjalan dengan normal. Bot mencari diamond terdekat dengan value terbesar, dan ketika kapasitas sudah maksimal bot langsung menuju ke base untuk menyetor hasil diamond yang dibawa.



Di pengujian kedua bot masih berjalan dengan normal dan konsisten. Tapi bot masih belum memanfaatkan fitur teleport. Dalam kondisi ini ketika kapasitas maksimal, bot yang seharusnya dapat memanfaatkan teleport untuk mempercepat perjalanan menuju base, bot malah langsung menuju base ketika kapasitas maksimal.



Pengujian ketiga ini bot mulai mengalami kesalahan ketika ingin kembali kebase. Kasusnya ketika bot sudah kapasitas maksimal dan ingin menuju ke base, namun di antara bot dan base terdapat teleport. Bot yang seharusnya menghindari teleport, malah tidak memperdulikannya dan pada akhirnya bot berpindah yang membuat jarak antara bot dan base lebih jauh.



Pada pengujian keempat, bot berjalan tidak sesuai harapan. Ketika kapasitas adalah 4, seharusnya bot memprioritaskan diamond dengan value terkecil bukan yang terdekat lagi. Yang terjadi ketika bot memaksa mengambil diamond yang akan melebihi kapasitas adalah bot hanya bolak-balik ke satu posisi sebelum diamond tersebut berada hingga waktu habis.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulannya adalah bot ini bergerak dengan tujuan mengumpulkan diamond hingga mencapai kapasistas(5), lalu membawa ke base untuk setor hasilnya. Jika kapasitas penuh atau tidak ada diamond yang tersedia, bot langsung diarahkan menuju base. Bot juga dapat menggunakan teleportasi jika itu lebih efisien dalam jarak ke diamond. Pemilihan arah gerak dilakukan secara strategis dengan mencari diamond terdekat dan diamond value terbanyak dan menghindari gerakan diagonal.

5.2 Saran

Saran pengembangan dari bot adalah untuk bisa memanfaatkan fitur reset board/diamond dengan memanfaatkan fitur itu bot akan bisa mendapatkan diamond lebih banyak.

LAMPIRAN

A. Repository Github

(https://github.com/RichRyanX/Tugas-Besar-Strategi-Algoritma-2025)

DAFTAR PUSTAKA

[1] “Algoritma Greedy: Pengertian, Jenis dan Contoh Program - FIKTI,” *Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Muhammadiyah Sumatera Utara (FIKTI-UMSU)*. [Online]. Available: https://fikti.umsu.ac.id/algoritma-greedy-pengertian-jenis-dan-contoh-program/ [Accessed: 31-May-2025].