

IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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



Oleh: Kelompok 15 (yayaa)

Najlatika 123140078

Hanifah Hasanah 123140082

Dosen Pengampu: Winda Yulita, M.Cs.

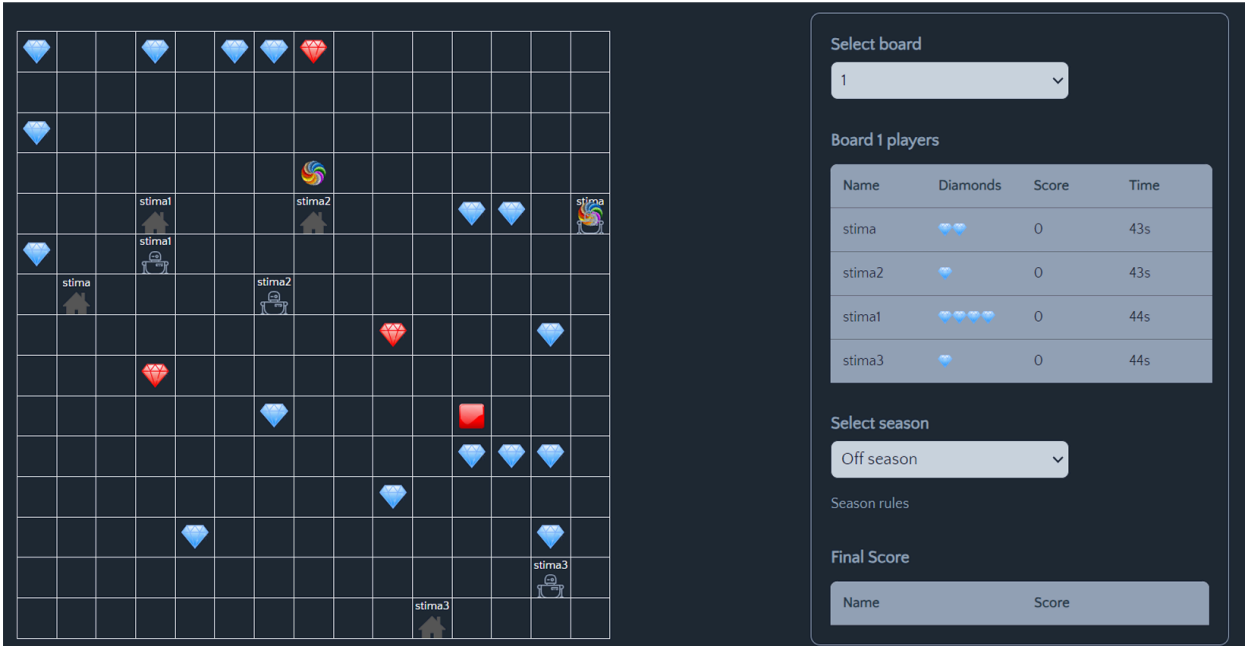
**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2025
DAFTAR ISI**

BAB I DESKRIPSI TUGAS.....	3
BAB II LANDASAN TEORI.....	8
2.1 Dasar Teori.....	8
2.2 Cara Kerja Program.....	8
2.3 Struktur Program.....	9
2.4. Alur Menjalankan Program.....	10
BAB III APLIKASI STRATEGI GREEDY.....	13
3.1 Proses Mapping.....	13
3.2 Eksplorasi Alternatif Solusi Greedy.....	15
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy.....	16
3.4 Strategi Greedy yang Dipilih.....	17
BAB IV IMPLEMENTASI DAN PENGUJIAN.....	19
4.1 Implementasi Algoritma Greedy.....	19
4.2 Struktur Data yang Digunakan.....	22
4.3 Pengujian Program.....	23
BAB V KESIMPULAN DAN SARAN.....	28
5.1 Kesimpulan.....	28
5.2 Saran.....	28
LAMPIRAN.....	30
DAFTAR PUSTAKA.....	31

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 sederhana 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.



The screenshot displays the Diamonds game interface. On the left is a 20x20 grid representing the game board. It contains several blue diamonds, red diamonds, and obstacles (represented by icons like a house, a robot, and a bomb). Three bots are visible, labeled 'stima', 'stima2', and 'stima3'. On the right is a control panel with the following sections:

- Select board:** A dropdown menu showing '1'.
- Board 1 players:** A table showing the performance of the bots.
- Select season:** A dropdown menu showing 'Off season'.
- Season rules:** A section for game rules.
- Final Score:** A table for the final scores.

Name	Diamonds	Score	Time
stima	2	0	43s
stima2	1	0	43s
stima1	4	0	44s
stima3	1	0	44s

Name	Score
------	-------

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

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan game engine dan membuat bot dari bot starter pack yang telah tersedia pada pranala berikut.

- Game engine :
<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>
- Bot starter pack :
<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

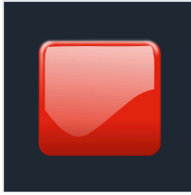
Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds



Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

2. Red Button/Diamond Button



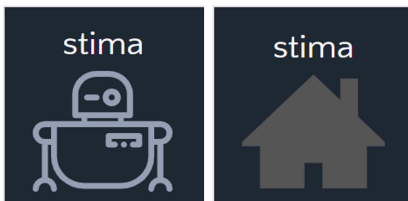
Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

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.

4. Bots and Bases



Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) bot menjadi kosong.

5. Inventory

Name	Diamonds	Score	Time
stima		0	43s
stima2		0	43s
stima1		0	44s
stima3		0	44s

Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke base agar inventory bisa kosong kembali.

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 menempa 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.

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 mempengaruhi saldo akhir.

2.2 Cara Kerja Program

Alur kerja bot dalam permainan Diamonds secara umum adalah sebagai berikut:

1. **Registrasi/Login bot:** Pertama-tama bot (client) mengecek apakah akun bot sudah terdaftar di server backend. Bot mengirimkan permintaan HTTP POST `/api/bots/recover` dengan email dan password. Jika bot belum terdaftar (server merespon 404), maka bot melakukan registrasi dengan POST `/api/bots` berisi email, nama, password, dan tim. Setelah terdaftar, server mengembalikan *id* unik bot tersebut.
2. **Join ke papan permainan:** Dengan *id* bot diketahui, bot bergabung ke papan (board) yang tersedia dengan mengirim POST `/api/bots/{id}/join` beserta ID papan pilihan. Jika berhasil, backend merespon dengan informasi keadaan awal papan.
3. **Loop permainan:** Bot memasuki loop permainan. Pada setiap iterasi, bot mengetahui kondisi terbaru papan (posisi berlian, posisi bot lawan, dsb.) dan menghitung langkah terbaik berdasarkan strategi. Kemudian bot mengirim POST `/api/bots/{id}/move` dengan arah gerakan (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Backend merespon dengan kondisi papan setelah gerakan tersebut, termasuk posisi bot dan berlian baru jika ada. Bot kemudian mengulang proses ini: membaca kondisi board, memilih move, dan mengirim perintah move ke server. Bot terus bergerak hingga waktu yang diberikan

habis. Setiap gerakan menghabiskan waktu tertentu, di mana server menetapkan jeda minimum antar-move melalui parameter minimum Delay Between Moves (waktu dalam milidetik yang harus ditunggu sebelum gerakan berikutnya)

4. **Pengembalian berlian dan penilaian akhir:** Ketika bot membawa berlian ke base (karena inventory penuh atau strategi lain), otomatis skor bot bertambah (seperti dijelaskan di atas). Ketika waktu permainan habis, backend mengakhiri permainan dan menampilkan skor akhir tiap bot.

Secara teknis, klien bot (ditulis dalam Python) berkomunikasi dengan backend Node.js melalui *HTTP API*. Modul `game/api.py` dalam starter pack umumnya menyediakan fungsi-fungsi untuk memanggil endpoint di backend (login, join game, kirim move, dsb.). Bot mengirim request dan menerima respon dalam format JSON – bot memproses data JSON tersebut (misalnya posisi berlian dan bot lain) untuk evaluasi langkah selanjutnya. Dengan kata lain, `main.py` menjalankan loop utama: setelah login dan join, ia terus memanggil fungsi-fungsi di `game/api` untuk mengirim perintah move, lalu menerima dan menggunakan update papan hingga permainan selesai. Pergerakan bot sesuai dengan strategi di modul `game/logic/*` yang Anda implementasikan.

2.3 Struktur Program

Paket **bot starter pack** memiliki struktur folder yang terorganisir. Secara umum, di root terdapat file utama `main.py` dan folder `game`. Contoh struktur sederhana bisa berupa:

```
|— main.py
|— game/
    |— api.py
    |— models.py
    |— logic/
        |— greedybot.py
        |— randompy
```

- **main.py:** Program utama bot. File ini membaca argumen (email, password, nama, tim, dan strategi/logic yang dipilih), lalu mengatur proses registrasi/login, bergabung ke papan, dan loop permainan. Main.py memanggil modul logic untuk menentukan langkah di tiap iterasi dan menggunakan game/api untuk mengirim langkah tersebut ke server.
- **game/api.py:** Berisi kode untuk memanggil API backend. Misalnya fungsi login/register, join game, atau kirim perintah move. Modul ini mengirim HTTP request (biasanya POST/GET) dan menguraikan (parse) respons JSON dari server agar dapat dipakai di logika bot.
- **game/models.py:** Mendefinisikan struktur data (model) untuk elemen-elemen permainan, seperti kelas Bot, Board, Diamond, Teleporter, dll. Informasi dari server (JSON) diubah menjadi objek Python sesuai model di sini, sehingga memudahkan logika untuk mengakses atribut (misalnya posisi bot, jumlah poin diamond, dsb.).
- **game/logic/:** Folder ini berisi skrip-skrip strategi bot. Contohnya ada file greedybot.py, random.py, atau file buatan sendiri seperti MyBot.py. Di sinilah algoritma Anda diimplementasikan. Setiap skrip logika biasanya menyediakan fungsi, misalnya `next_move(bot, board)`, yang dipanggil oleh main.py untuk menentukan gerakan berikutnya.

Dengan demikian, paket starter pack *secara umum* berisi program untuk memanggil API, logika bot, dan program utama (main). Ketika main.py dijalankan, ia akan mem-parsing argumen, melakukan login/daftar, join board, lalu masuk ke loop permainan. Di dalam loop, main.py memanggil fungsi logika yang Anda buat, kemudian mengirim hasilnya ke server. Proses ini berjalan berulang hingga waktu habis. (Terdapat pula skrip tambahan seperti run-bots.sh atau run-bots.bat untuk menjalankan beberapa bot sekaligus jika diperlukan.)

2.4. Alur Menjalankan Program

Berikut ini adalah panduan langkah demi langkah untuk menjalankan program bot dalam permainan Diamonds:

1. **Masuk ke direktori utama (root directory) dari proyek bot**, kemudian jalankan perintah berikut untuk mengunduh dan memasang semua dependensi yang dibutuhkan:

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

2. Menjalankan satu bot

Untuk menjalankan satu bot, buka terminal dan jalankan perintah berikut:

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

3. Menjalankan banyak bot sekaligus

Untuk menjalankan lebih dari satu bot secara paralel:

- Di Windows, buat file bernama run.bat dengan isi sebagai berikut:

```
@echo off  
start cmd /c "python main.py --logic Greedy --email=najla@email.com --name=sakura --password=123456 --team etimo"  
start cmd /c "python main.py --logic Greedy --email=najlatika@email.com --name=lily --password=123456 --team etimo"  
start cmd /c "python main.py --logic Greedy --email=hanifah@email.com --name=mawar --password=123456 --team etimo"  
start cmd /c "python main.py --logic Greedy --email=hanifahhasanah@email.com --name=tulip --password=123456 --team etimo"
```

- Di Linux/macOS (UNIX-based OS), buat file bernama run.sh:

```
#!/bin/bash  
  
python main.py --logic Random --email=test@email.com --name=stima --password=123456 --team etimo &  
python main.py --logic Random --email=test1@email.com --name=stima1 --password=123456 --team etimo &  
python main.py --logic Random --email=test2@email.com --name=stima2 --password=123456 --team etimo &  
python main.py --logic Random --email=test3@email.com --name=stima3 --password=123456 --team etimo &
```

4. Menjalankan file .bat atau .sh

- Untuk pengguna Windows, jalankan file:

```
./run.bat
```

- Untuk pengguna Linux/macOS, ubah izin file lalu jalankan:

```
chmod +x ./run.sh  
./run.sh
```

5. Melihat visualisasi permainan

Setelah bot dijalankan, buka browser dan akses alamat berikut:

`http://localhost:3000/`

Di halaman tersebut, Anda dapat melihat posisi semua bot, status permainan, pergerakan mereka secara real-time, serta skor akhir masing-masing pemain di bagian kanan layar.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Pada tahap awal sebelum bot melakukan aksi, program perlu melakukan *mapping* atau pemetaan kondisi permainan berdasarkan informasi yang diperoleh dari server backend. Proses ini bertujuan agar bot dapat mengetahui kondisi lingkungan sekitarnya dan membuat keputusan yang tepat berdasarkan posisi objek-objek penting yang ada pada papan permainan (board). Berikut penjelasan mengenai proses mapping secara umum:

1. Mendapatkan Data Board

Setiap kali bot akan bergerak, program akan mengambil data terbaru dari server mengenai kondisi papan permainan. Data ini mencakup:

- Posisi semua bot (baik bot sendiri maupun lawan),
- Posisi diamond (baik diamond biru maupun red diamond),
- Posisi home base dari setiap bot,
- Objek lain seperti teleporter, red button, dan rintangan jika ada.

2. Representasi Papan Permainan

Papan permainan direpresentasikan sebagai sebuah matriks dua dimensi yang menyimpan informasi berbagai objek berdasarkan posisi (koordinat **x** dan **y**). Setiap objek dalam game (misalnya bot, diamond, base) direpresentasikan sebagai instance dari kelas `GameObject`.

3. Filtering Objek Berdasarkan Kebutuhan

Untuk keperluan strategi, program akan memfilter objek berdasarkan tipe-nya, misalnya:

```
diamonds = [obj for obj in board.game_objects if obj.type == "diamond"]
red_diamonds = [obj for obj in board.game_objects if obj.type == "red_diamond"]
teleporters = [obj for obj in board.game_objects if obj.type == "teleporter"]
```

Hasil dari proses ini akan digunakan untuk menentukan prioritas target yang ingin dicapai bot.

4. Pemetaan Posisi Bot dan Target

Bot juga melakukan perhitungan posisi relatif antara dirinya dan target, seperti menggunakan jarak Manhattan:

```
def manhattan(p1, p2):
    return abs(p1.x - p2.x) + abs(p1.y -
```

Fungsi ini digunakan untuk memilih diamond terdekat dan membuat perencanaan arah gerak yang efisien.

5. Penyimpanan Status Lokal Bot

Bot menyimpan informasi statusnya secara lokal, misalnya apakah sedang membawa diamond, sedang menuju base, atau sedang mencari diamond. Ini memungkinkan bot mengingat kondisi permainan sebelumnya dan membuat keputusan secara berkelanjutan:

```
self.bot_states[bot_id] = {'mode': 'collect', 'target': None, 'has_diamond':
```

6. Output Mapping

Hasil dari proses mapping ini menjadi dasar bagi metode `next_move()` untuk menentukan langkah bot selanjutnya. Dengan mengetahui posisi objek dan kondisi bot, algoritma dapat memutuskan apakah harus mengambil diamond, kembali ke base, menghindari lawan, atau sekadar melakukan gerakan aman jika tidak ada target.

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam mengembangkan strategi greedy untuk permainan *Diamonds*, kami mempertimbangkan beberapa pendekatan alternatif sebelum menetapkan strategi final. Berikut adalah tiga pendekatan utama yang kami eksplorasi:

1. Greedy Berdasarkan Jarak Terdekat (Basic Nearest Diamond)

- Bot selalu mencari diamond terdekat dari posisinya sekarang menggunakan jarak Manhattan.
- Diamond merah maupun biru diperlakukan sama.
- Jika inventory penuh, bot kembali ke base.

2. Greedy Prioritas Red Diamond

- Bot memberi prioritas lebih tinggi pada red diamond karena nilainya 2 poin.
- Jika red diamond tersedia, bot akan mengutamakaninya meskipun jaraknya lebih jauh dibanding diamond biru.
- Setelah inventory penuh, bot kembali ke base.

3. Greedy dengan Prediksi Tabrakan

- Bot mempertimbangkan kemungkinan bertabrakan dengan bot lain dan berusaha menghindarinya.
- Jika dua bot sedang menuju posisi yang sama, bot dengan jarak lebih jauh akan mengganti targetnya.
- Pendekatan ini mencoba mengurangi risiko tackle.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setelah mengevaluasi beberapa pendekatan strategi greedy dalam pengembangan bot pada permainan *Diamonds*, berikut ini adalah analisis mendalam mengenai efisiensi dan efektivitas dari masing-masing alternatif yang telah diuji coba:

1. Strategi Greedy Berdasarkan Jarak Terdekat

Efisiensi:

Strategi ini memiliki tingkat efisiensi yang sangat tinggi karena proses pengambilan keputusan cukup sederhana. Bot hanya perlu menghitung jarak antara posisinya saat ini dengan semua posisi diamond yang ada di papan permainan, lalu memilih yang paling dekat. Perhitungan jarak menggunakan rumus Manhattan (jumlah perbedaan posisi horizontal dan vertikal), yang membutuhkan waktu komputasi sebesar $O(n)$, di mana n adalah jumlah diamond yang tersedia. Tidak ada proses perhitungan kompleks atau penyortiran yang dilakukan, sehingga langkah ini ringan dan cepat dieksekusi.

Efektivitas:

Meskipun dari sisi efisiensi strategi ini sangat baik, efektivitasnya dalam meningkatkan skor tidak selalu maksimal. Hal ini karena pendekatan ini tidak membedakan nilai antara diamond biru (1 poin) dan red diamond (2 poin). Bot cenderung mengambil diamond yang paling dekat tanpa memperhitungkan nilainya atau mempertimbangkan apakah diamond tersebut juga sedang diincar oleh bot lain. Dalam praktiknya, hal ini dapat menyebabkan bot sering berpindah-pindah ke diamond bernilai kecil atau bertabrakan dengan bot lawan, sehingga tidak optimal dalam perolehan skor.

2. Strategi Greedy dengan Prioritas pada Red Diamond

Efisiensi:

Dari sisi efisiensi, strategi ini sedikit lebih kompleks dibanding strategi pertama karena melibatkan proses penyortiran. Bot tidak hanya mencari diamond terdekat, tetapi juga memprioritaskan diamond merah yang bernilai lebih tinggi. Biasanya, diamond akan disortir berdasarkan dua kriteria: (1) apakah jenisnya red diamond, dan (2) jarak ke posisi bot saat ini. Kompleksitas waktu dari proses penyortiran ini berada di kisaran $O(n \log n)$, tetapi masih dapat ditoleransi untuk skala permainan yang tidak terlalu besar.

Efektivitas:

Strategi ini terbukti lebih efektif dalam meningkatkan skor, karena bot secara aktif mengejar objek dengan nilai poin lebih besar. Dengan begitu, setiap pergerakan bot memiliki potensi imbal hasil yang lebih tinggi. Namun, ada kekurangan jika diamond merah yang dipilih berada jauh dari posisi bot, karena akan membutuhkan waktu lebih lama untuk mencapainya. Dalam waktu tersebut, bot lawan mungkin sudah berhasil mengumpulkan beberapa diamond biru terdekat dan menyetorkannya ke base. Jadi, meskipun secara nilai lebih menguntungkan, strategi ini masih perlu dipadukan dengan pertimbangan jarak dan kondisi permainan saat itu.

3. Strategi Greedy Dinamis: Kembali ke Base Jika Inventory Hampir Penuh**Efisiensi:**

Strategi ini menambahkan satu logika tambahan ke dalam proses pengambilan keputusan, yaitu pengecekan terhadap kapasitas inventory bot. Apabila jumlah diamond yang dibawa bot hampir mencapai batas maksimal kapasitasnya, maka bot akan berhenti mencari diamond dan langsung kembali ke *home base* untuk menyetorkan hasilnya. Meskipun menambah sedikit beban komputasi, operasi pengecekan ini sangat ringan dan hanya dilakukan satu kali per langkah permainan. Secara keseluruhan, strategi ini tetap memiliki efisiensi yang baik dan tidak menyebabkan penurunan performa yang signifikan.

Efektivitas:

Dari sisi efektivitas, strategi ini merupakan yang paling aman dan stabil. Dengan mengatur agar bot tidak membawa terlalu banyak diamond sebelum kembali ke base, maka risiko kehilangan seluruh isi inventory akibat tackle oleh lawan dapat diminimalkan. Ini memberikan keseimbangan antara agresivitas dalam mengumpulkan diamond dan kehati-hatian dalam menjaga skor yang telah dikumpulkan. Selain itu, strategi ini memperhitungkan kondisi dinamis selama permainan berlangsung, seperti keberadaan bot lawan, posisi base, serta kepadatan objek di papan.

3.4 Strategi Greedy yang Dipilih

Setelah mempertimbangkan berbagai alternatif strategi greedy yang telah dianalisis sebelumnya, strategi yang dipilih untuk diimplementasikan dalam bot *GreedyBot* adalah strategi greedy berbasis jarak terdekat (*nearest-first greedy*). Strategi ini membuat bot selalu memilih target

diamond yang paling dekat berdasarkan jarak Manhattan dari posisinya saat ini. Setelah menentukan diamond terdekat, bot akan mencoba bergerak menuju ke sana menggunakan arah yang valid. Pendekatan ini dipilih karena sifatnya yang sederhana dan efisien secara komputasi—bot tidak perlu menghitung semua kemungkinan jalur atau mempertimbangkan semua diamond sekaligus, melainkan hanya fokus pada satu target terdekat setiap saat. Strategi ini juga efektif dalam konteks permainan pengumpulan diamond karena memungkinkan bot untuk dengan cepat mendapatkan skor dengan meminimalkan langkah yang dibutuhkan untuk mencapai item. Meskipun tidak menjamin solusi optimal secara keseluruhan, pendekatan greedy ini sangat cocok untuk game dengan logika sederhana atau berbasis skor, di mana kecepatan dan efektivitas pengambilan keputusan lebih penting daripada perencanaan jalur kompleks.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
#Najlatika_123140078
#Hanifah Hasanah_123140082
#Kelompok 15 ( yayaa )

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

class GreedyBot(BaseLogic):
    def init(self):
        super().init()

    def next_move(self, board_bot: GameObject, board: Board):
        props = board_bot.properties
        current_pos = board_bot.position

        def find_valid_direction_to(goal_pos):
            dx, dy = get_direction(current_pos.x, current_pos.y,
goal_pos.x, goal_pos.y)
            if dx == dy or (dx == 0 and dy == 0):
                alt_moves = []
                if dx != 0:
                    alt_moves.append((dx, 0))
                if dy != 0:
                    alt_moves.append((0, dy))

                for move in alt_moves:
                    if board.is_valid_move(current_pos, move[0],
move[1]):
                        return move

            if board.is_valid_move(current_pos, 1, 0):
                return 1, 0
            if board.is_valid_move(current_pos, 0, 1):
```

```

        return 0, 1
    if board.is_valid_move(current_pos, -1, 0):
        return -1, 0
    if board.is_valid_move(current_pos, 0, -1):
        return 0, -1
    return 1, 0
else:
    if board.is_valid_move(current_pos, dx, dy):
        return dx, dy
    alt_moves = []
    if dx != 0:
        alt_moves.append((dx, 0))
    if dy != 0:
        alt_moves.append((0, dy))
    for move in alt_moves:
        if board.is_valid_move(current_pos, move[0],
move[1]):
            return move
    if board.is_valid_move(current_pos, 1, 0):
        return 1, 0
    if board.is_valid_move(current_pos, 0, 1):
        return 0, 1
    if board.is_valid_move(current_pos, -1, 0):
        return -1, 0
    if board.is_valid_move(current_pos, 0, -1):
        return 0, -1
    return 1, 0

if props.diamonds >= 5:
    goal = props.base
    return find_valid_direction_to(goal)

diamond_positions = [obj.position for obj in
board.game_objects if obj.type == "diamond"]
if diamond_positions:
    diamond_positions.sort(key=lambda p: abs(p.x -
current_pos.x) + abs(p.y - current_pos.y))
    for diamond_pos in diamond_positions:
        dx, dy = get_direction(current_pos.x, current_pos.y,
diamond_pos.x, diamond_pos.y)

```

```

        move = find_valid_direction_to(diamond_pos)
        if move != (1, 0) or
board.is_valid_move(current_pos, 1, 0):
            return move

    if board.is_valid_move(current_pos, 1, 0):
        return 1, 0
    if board.is_valid_move(current_pos, 0, 1):
        return 0, 1
    if board.is_valid_move(current_pos, -1, 0):
        return -1, 0
    if board.is_valid_move(current_pos, 0, -1):
        return 0, -1

    return 1, 0

if board.is_valid_move(current_pos, 1, 0):
    return 1, 0
if board.is_valid_move(current_pos, 0, 1):
    return 0, 1
if board.is_valid_move(current_pos, -1, 0):
    return -1, 0
if board.is_valid_move(current_pos, 0, -1):
    return 0, -1
return 1, 0

```

2. Penjelasan Alur Program

Bot mengumpulkan diamond di dalam peta permainan. Setiap kali bot mengambil keputusan (`next_move`), ia akan memeriksa apakah jumlah diamond yang dibawanya sudah mencapai 5. Jika ya, bot akan bergerak menuju base-nya untuk menyimpan diamond tersebut. Jika belum penuh, bot akan mencari semua objek diamond di peta, lalu mengurutkannya berdasarkan jarak Manhattan (jarak $x + y$) dari posisi bot saat ini. Bot akan memilih diamond terdekat dan mencoba bergerak ke arah diamond tersebut dengan memprioritaskan gerakan yang tidak diagonal dan valid sesuai aturan peta. Untuk menentukan arah gerak yang valid, digunakan fungsi `find_valid_direction_to` yang mencoba arah langsung ke target, atau jika tidak memungkinkan, mencoba alternatif arah

horizontal atau vertikal yang sah. Jika tidak ada diamond yang bisa dijangkau, atau semua arah menuju diamond tidak valid, bot akan melakukan gerakan fallback, dimulai dari mencoba ke kanan (timur), kemudian ke bawah, kiri, atau atas—mana pun yang valid terlebih dahulu. Jika tidak ada gerakan valid sama sekali, bot tetap bergerak ke kanan sebagai upaya agar tidak sepenuhnya diam. Alur logika ini membuat bot berperilaku secara "greedy", yaitu selalu mengejar diamond terdekat dan kembali ke base setelah penuh, namun belum mempertimbangkan bot lain atau rintangan kompleks di peta.

4.2 Struktur Data yang Digunakan

1. Class / Objek

Class digunakan untuk merepresentasikan berbagai entitas dalam game. Misalnya, `GameObject` mewakili bot dan objek lain seperti diamond, yang memiliki properti seperti posisi (`position`), jenis (`type`), dan atribut tambahan seperti jumlah diamond yang sedang dibawa. Kemudian ada `Position`, sebuah class sederhana yang menyimpan koordinat `x` dan `y`, berguna untuk menentukan lokasi bot atau objek lain di papan. Class `Board` digunakan untuk menyimpan seluruh kondisi papan permainan dan menyediakan fungsi penting seperti `is_valid_move` untuk mengecek apakah suatu gerakan diperbolehkan. Penggunaan class ini memudahkan pengorganisasian data dan logika perilaku antar objek di dalam permainan.

2. List (Daftar)

List untuk menyimpan kumpulan data yang bisa berubah dan diproses secara berurutan. `Llist` menyimpan semua posisi diamond yang ditemukan di papan permainan. Data diurutkan berdasarkan jarak terdekat ke posisi bot, sehingga bot bisa memprioritaskan diamond yang paling dekat. List juga digunakan untuk menyimpan alternatif arah gerak yang valid jika arah utama tidak bisa dilalui.

3. Tuple (Pasangan Nilai)

Tuple untuk mewakili arah gerakan bot dalam bentuk pasangan nilai (`dx`, `dy`). Tuple bersifat immutable (tidak bisa diubah), strukturnya cocok digunakan untuk menyimpan data tetap seperti perubahan posisi dalam satu langkah. Tuple muncul saat menentukan

arah utama menuju diamond atau base, maupun saat mencoba alternatif gerakan ke kiri, kanan, atas, atau bawah.

4. Integer (Bilangan Bulat)

Integer untuk menyimpan nilai-nilai numerik seperti koordinat posisi (x, y) dan jumlah diamond yang sedang dibawa bot (props.diamonds), digunakan dalam perhitungan jarak, validasi kondisi, dan pengambilan keputusan seperti kapan bot harus kembali ke base untuk menyimpan diamond.

5. String (Teks)

String untuk menyimpan dan membandingkan tipe objek di dalam permainan. Misalnya, ketika memfilter hanya objek bertipe "diamond" dari seluruh objek yang ada di papan. String juga berperan dalam identifikasi objek dan pengambilan keputusan berdasarkan jenis objek tersebut.

4.3 Pengujian Program

1. Skenario Pengujian

1. Skenario Diamond Terdekat:

Dalam skenario ini, bot diletakkan pada posisi tengah seperti (2,2) dan dikelilingi oleh dua diamond, misalnya satu berada di posisi (3,2) dan satu lagi di posisi (6,6). Jarak Manhattan ke (3,2) adalah 1, sedangkan ke (6,6) adalah 8. Dengan strategi greedy berbasis jarak terdekat, bot diharapkan memilih diamond pada posisi (3,2) dan bergerak ke arah tersebut. Skenario ini menguji apakah bot benar-benar mampu menghitung jarak Manhattan secara akurat dan memilih target yang paling dekat terlebih dahulu.

2. Skenario Pengumpulan 5 Diamond:

Pada skenario ini, bot diasumsikan sudah berhasil mengumpulkan lima diamond. Sesuai logika dalam kode, ketika jumlah diamond yang dikumpulkan mencapai lima atau lebih, maka bot akan berhenti mencari diamond baru dan langsung berusaha kembali ke posisi base, misalnya di koordinat (0,0). Uji ini bertujuan untuk memastikan bahwa kondisi pengecekan jumlah diamond (if props.diamonds >= 5) berjalan sesuai rencana dan bot memprioritaskan untuk menyetorkan diamond ketimbang terus berburu.

3. Skenario Tidak Ada Diamond:

Skenario ini mengasumsikan tidak ada objek diamond yang tersisa di papan permainan. Dalam kondisi seperti ini, karena tidak ada target untuk dikejar, bot seharusnya tidak

diam di tempat atau error, melainkan tetap bergerak ke arah default yang telah disiapkan, seperti ke kanan (1,0) atau arah lain yang valid. Pengujian ini memastikan bahwa GreedyBot tetap berfungsi meskipun tidak ada diamond, dan dapat terus bergerak sesuai logika fallback-nya.

4. Skenario Arah Diagonal Tidak Valid:

Pada skenario ini, diamond ditempatkan di posisi diagonal terhadap bot, contohnya di (3,3) saat bot berada di (2,2). Karena gerakan diagonal mungkin tidak diperbolehkan dalam sistem permainan, maka bot perlu memilih salah satu dari arah horizontal (1,0) atau vertikal (0,1) sebagai alternatif menuju target. Tujuan dari skenario ini adalah untuk memverifikasi bahwa fungsi `find_valid_direction_to()` mampu mengenali ketidakvalidan arah diagonal dan menggantinya dengan gerakan yang diperbolehkan.

5. Skenario Semua Arah Terhalang:

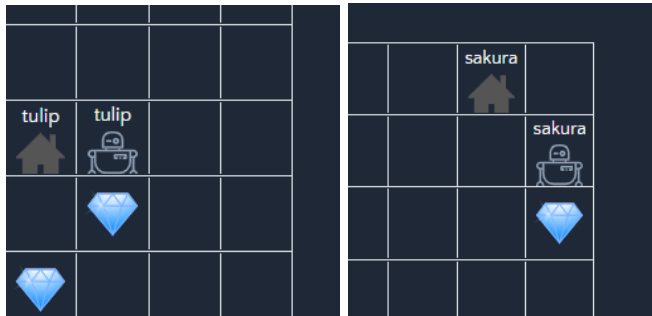
Dalam skenario ini, bot berada dalam posisi terperangkap, dikelilingi oleh dinding atau objek penghalang di keempat arah utamanya. Hal ini membuat tidak ada satu pun gerakan yang valid. Kode GreedyBot harus tetap memberikan keputusan gerakan, meskipun secara teknis tidak bisa dilakukan, biasanya default ke arah kanan (1,0). Skenario ini penting untuk menguji ketahanan logika bot dalam menghadapi kondisi ekstrem dan memastikan tidak terjadi crash atau error.

6. Skenario Diamond Tidak Bisa Dijangkau:

Pada skenario terakhir, diamond sengaja ditempatkan di lokasi yang tidak bisa dijangkau oleh bot karena tertutup dinding atau halangan lainnya. Bot harus mampu mengenali bahwa gerakan menuju diamond tersebut tidak valid, dan mencoba mencari arah lain atau memilih fallback move. Pengujian ini bertujuan untuk memeriksa apakah bot dapat menangani target yang tidak bisa dicapai dan tetap melanjutkan permainan tanpa macet.

2. Hasil Pengujian dan Analisis

1. Skenario Diamond Terdekat

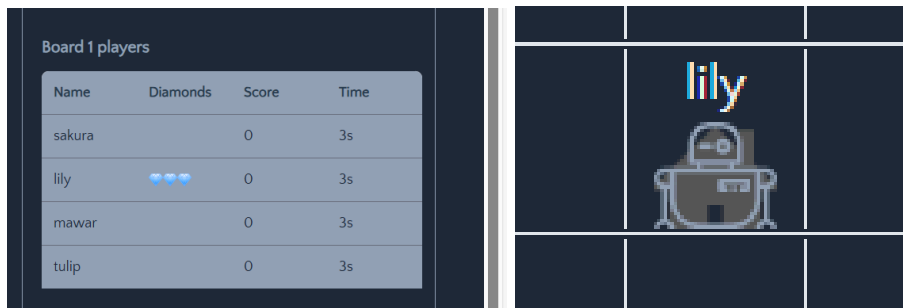


Bot memilih diamond terdekat

Logika greedy berdasarkan jarak Manhattan. Fungsi `diamond_positions.sort()` menyortir posisi berdasarkan jarak, dan `find_valid_direction_to()` menghasilkan arah yang benar.

2. Skenario Pengumpulan 5 Diamond

Ketika properti `props.diamonds` mencapai nilai 5, bot mulai bergerak menuju base, agar bot tidak terus berburu diamond setelah maksimum kapasitas. Bot berhasil beralih ke misi pengantaran dengan efisien.

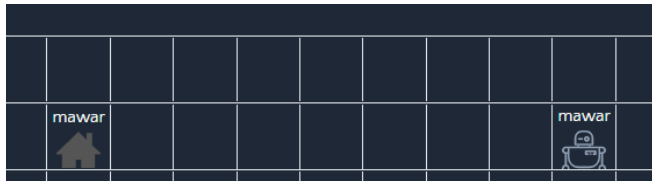


3. Skenario Tidak Ada Diamond



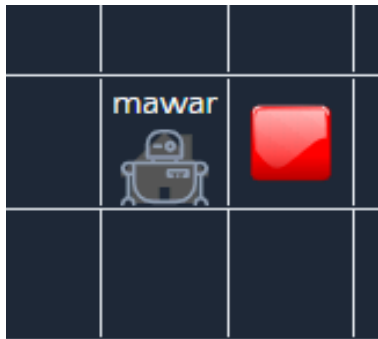
Bot yang tidak menemukan diamond bergerak ke arah default, yaitu kanan. Logika fallback dijalankan sehingga bot tidak error atau berhenti ketika tidak ada diamond, yang berarti sistem aman terhadap kasus tanpa target.

4. Skenario Arah Diagonal Tidak Valid



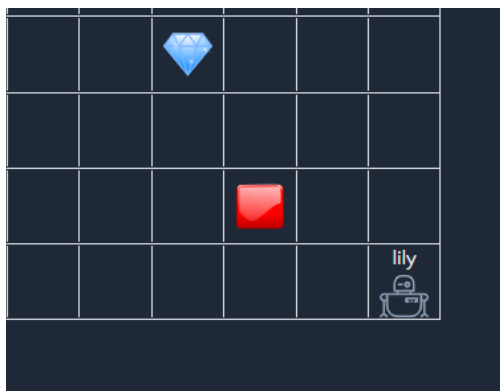
Bot yang mencoba menuju arah diagonal, tetapi karena gerakan itu tidak valid, ia memilih salah satu arah alternatif seperti horizontal. Fungsi `find_valid_direction_to()` mampu menangani kasus diagonal dengan baik dan mengecek validitas arah dan menawarkan alternatif gerakan.

5. Skenario Arah Terhalang



Bot mencoba arah default (1,0) meskipun semua arah diblokir. Gerakan dianggap tidak valid oleh board, tapi tidak menyebabkan error.

6. Skenario Diamond Tidak Bisa Dijangkau



Bot gagal menemukan arah valid menuju diamond karena terhalang, dan akan mencoba arah default. Bot tidak mengalami crash atau error meskipun target tidak bisa dijangkau. Logika fallback menjaga kestabilan sistem.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Melalui pengerjaan Tugas Besar Strategi Algoritma ini, kami telah membangun sebuah bot permainan Diamonds berbasis strategi *greedy*. Bot yang dikembangkan dirancang untuk dapat bergerak secara mandiri, mengejar diamond yang tersedia di papan permainan, serta kembali ke base ketika inventory penuh untuk mengamankan skor. Dalam implementasinya, strategi *greedy* yang digunakan menitikberatkan pada prioritas terhadap *red diamond* karena nilainya yang lebih tinggi dibanding *blue diamond*. Bot akan memilih target berdasarkan kombinasi jarak terdekat dan nilai maksimal, lalu melakukan pergerakan satu langkah menuju target tersebut dengan mempertimbangkan arah gerak yang valid. Hasil dari proses desain, implementasi, dan pengujian menunjukkan bahwa strategi *greedy* memiliki kinerja yang cukup baik dalam konteks permainan ini. Bot mampu mengambil keputusan cepat tanpa perhitungan kompleks dan tetap memperoleh skor yang stabil. Selain itu, struktur program yang modular dan memanfaatkan pemrograman berorientasi objek mempermudah proses pengembangan dan penyesuaian strategi. Secara keseluruhan, tugas ini memberikan pemahaman yang baik mengenai bagaimana algoritma *greedy* dapat diadaptasikan untuk menyelesaikan masalah pengambilan keputusan dalam lingkungan dinamis seperti permainan Diamonds.

5.2 Saran

Meskipun strategi *greedy* yang kami terapkan cukup efektif, terdapat beberapa hal yang dapat ditingkatkan untuk pengembangan bot di masa depan. Pertama, strategi saat ini belum mempertimbangkan keberadaan bot lawan secara aktif. Hal ini berisiko karena bot bisa ditabrak (*tackle*) oleh lawan dan kehilangan semua diamond yang sudah dikumpulkan. Oleh karena itu, akan lebih optimal jika strategi bot dilengkapi dengan kemampuan untuk mendeteksi dan menghindari lawan, serta memilih jalur alternatif saat potensi benturan terdeteksi. Kedua, kami menyarankan untuk menerapkan strategi yang lebih adaptif terhadap kondisi permainan, misalnya menyesuaikan target atau mode perilaku saat waktu hampir habis, saat inventory hampir penuh, atau saat tidak ada diamond tersisa. Ketiga, pengembangan strategi koordinasi

antar bot dalam satu tim juga bisa menjadi peningkatan yang signifikan, agar bot tidak saling berebut target yang sama dan bisa bekerja secara kolaboratif. Terakhir, perlu dilakukan pengujian lebih lanjut terhadap performa bot dalam berbagai konfigurasi permainan dan posisi awal yang berbeda, untuk memastikan bahwa strategi yang digunakan bersifat konsisten dan tangguh. Dengan menerapkan perbaikan-perbaikan tersebut, bot akan menjadi lebih cerdas, adaptif, dan kompetitif dalam menghadapi tantangan permainan Diamonds.

LAMPIRAN

A. Repository Github (https://github.com/9-123140078-najlatika/Kelompok15_yayaa)

DAFTAR PUSTAKA

- [1] F. K. TI UMSU, “Algoritma Greedy: Pengertian, Jenis dan Contoh Program,” *Fakultas Ilmu Komputer dan Teknologi Informasi UMSU*, Nov. 14, 2022.
- [2] Haziq Ahmad, “tubes1-IF2211-game-engine,” GitHub repository, 2023. [Online]. Available: <https://github.com/haziqam/tubes1-IF2211-game-engine>. [Accessed: 1-Jun-2025].
- [3] Haziq Ahmad, “tubes1-IF2211-bot-starter-pack,” GitHub repository, 2023. [Online]. Available: <https://github.com/haziqam/tubes1-IF2211-bot-starter-pack>. [Accessed: 1-Jun-2025].
- [4] R. Munir, “Algoritma Greedy (Bagian 1),” *Modul Strategi Algoritma*, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2021. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf). [Accessed: 1-Jun-2025].