LAPORAN TUGAS BESAR IF 2211
STRATEGI ALGORITMA

“Pemanfaatan Algoritma Greedy dalam pembuatan bot permainan Diamonds”

Oleh: KELOMPOK ELEVENTWELFTH

Devina Kartika	123140036
Muhammad Nurikhsan	123140057
Aryasatya Widyanta Akbar	123140164

Dosen Pengampu: Winda Yulita, M.Cs.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I **2**
DESKRIPSI TUGAS 2
BAB II **8**
LANDASAN TEORI 8
2.1 Dasar Teori 8
2.2 Cara Kerja Program 8
1. Implementasi Algoritma Greedy ke dalam Bot 8
2. Cara Menjalankan Bot 9
3. Penanganan Khusus 9
BAB III **10**
APLIKASI STRATEGI GREEDY 10
3.1 Proses Mapping 10
3.2 Eksplorasi Alternatif Solusi Greedy 11
3.3 Analisis Efisiensi dan Efektivitas Alternatif Solusi Greedy 12
3.4 Strategi Greedy yang Dipilih 13
BAB IV **14**
IMPLEMENTASI DAN PENGUJIAN 14
4.1 Implementasi Algoritma Greedy 14
1. Pseudocode 14
2. Penjelasan Alur Program 19
4.2 Struktur Data yang Digunakan 20
4.3 Analisis Desain Solusi Algoritma 22
4.4 Pengujan Bot 23
1. Skenario pengujian 23
2. Hasil Pengujian dan Analisis 23
BAB V **27**
KESIMPULAN DAN SARAN 27
5. 1 Kesimpulan 27
5.2 Saran 27
LAMPIRAN **28**
A. Repository GitHub: 28
DAFTAR PUSTAKA **29**

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.

Gambar 1. Permainan Diamond

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 ermainan
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**

Gambar 2. Diamond Biru dan Merah

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**

Gambar 3. Red/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**

Gambar 4. 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**

Gambar 5. 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 dibawah) bot menjadi kosong.

5. **Inventory**

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

Gambar 6. Inventory

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 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.

Panduan Penggunaan

Adapun panduan mengenai cara instalasi, menjalankan permainan, membuat bot, melihat visualizer/frontend, dan mengatur konfigurasi permainan dapat dilihat melalui tautan berikut: https://docs.google.com/document/d/1L92Axb89yIkom0b24D350Z1QAr8rujvHof7-kXRAp7c

Mekanisme Teknis Permainan Diamonds

Permainan ini merupakan permainan berbasis web, sehingga setiap aksi yang dilakukan mulai dari mendaftarkan bot hingga menjalankan aksi bot – akan memerlukan HTTP request terhadap API endpoint tertentu yang disediakan oleh backend. Berikut adalah urutan requests yang terjadi dari awal mula permainan.

1. Program bot akan mengecek apakah bot sudah terdaftar atau belum, dengan mengirimkan POST request terhadap endpoint /api/bots/recover dengan body berisi email dan password bot. Jika bot sudah terdaftar, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut. Jika tidak, backend akan memberikan response code 404.
2. Jika bot belum terdaftar, maka program bot akan mengirimkan POST request terhadap endpoint /api/bots dengan body berisi email, name, password, dan team. Jika berhasil, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut.
3. Ketika id bot sudah diketahui, bot dapat bergabung ke board dengan mengirimkan POST request terhadap endpoint /api/bots/{id}/join dengan body berisi board id yang diinginkan (preferredBoardId). Apabila bot berhasil bergabung, maka backend akan memberikan response code 200 dengan body berisi informasi dari board.
4. Program bot akan mengkalkulasikan move selanjutnya secara berkala berdasarkan kondisi board yang diketahui, dan mengirimkan POST request terhadap endpoint /api/bots/{id}/move dengan body berisi direction yang akan ditempuh selanjutnya (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Apabila berhasil, maka backend akan

memberikan response code 200 dengan body berisi kondisi board setelah move tersebut. Langkah ini dilakukan terus-menerus hingga waktu bot habis. Jika waktu bot habis, bot secara otomatis akan dikeluarkan dari board.

5. Program frontend secara periodik juga akan mengirimkan GET request terhadap endpoint /api/boards/{id} untuk mendapatkan kondisi board terbaru, sehingga tampilan board pada frontend akan selalu ter-update.

Spesifikasi Tugas Besar 1

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

- Terdapat juga demo dari program yang telah dibuat. Pengumuman tentang demo menunggu pemberitahuan lebih lanjut dari asisten.
- Bot yang telah dibuat akan dikompetisikan dengan kelompok lain dan disaksikan oleh seluruh peserta kuliah. Terdapat hadiah menarik bagi kelompok yang memenangkan kompetisi.
- Setiap kelompok harap melaporkan nama kelompok dan anggotanya dan diserahkan kepada asisten untuk didata.
- Kelompok yang terindikasi melakukan kecurangan akan diberikan nilai 0 pada tugas besar.
- Program disimpan dalam repository yang bernama Tubes1 NamaKelompok dengan nama kelompok. Berikut merupakan struktur dari isi repository tersebut:
    a. Folder src berisi *source code.*
    b. Folder doc berisi laporan tugas besar dengan format NamaKelompok.pdf
    c. README untuk tata cara penggunaan yang minimal berisi:
        i. Penjelasan singkat algoritma greedy yang diimplementasikan
        ii. Requirement program dan instalasi tertentu bila ada
        iii. Command atau langkah-langkah dalam meng-compile atau build program
        iv. Author (identitas pembuat)
- Laporan dikumpulkan hari Minggu, 1 Juni 2025 pada alamat Google Form berikut paling lambat pukul 23.59 : https://bit.ly/4hmMMs0
- Adapun pertanyaan terkait tugas besar ini bisa disampaikan melalui QnA berikut: https://bit.ly/4iSjlPk

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma *Greedy* merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi, yang dapat dibagi menjadi dua macam persoalan, yaitu Maksimasi dan Minimasi. Algoritma *Greedy* ini digunakan untuk memecahkan masalah langkah demi langkah, di mana pada setiap langkah akan dipilih-pilihan yang memberikan keuntungan terbesar pada saat itu (*locally optimal choice*). Algoritma ini berharap bahwa dengan memilih pilihan optimal lokal pada setiap tahap, pada akhirnya akan mencapai solusi optimal global.

2.2 Cara Kerja Program

Program Bot dirancang untuk bermain secara otomatis dalam game *Diamonds* dengan membaca kondisi papan permainan secara *real-time.* Setiap giliran, bot menganalisis posisi dirinya, lokasi diamond, keberadaan lawan, serta objek lain seperti *base,* tombol, dan *teleporter.*

Dengan menggunakan algoritma *greedy*, bot akan memilih tujuan yang paling menguntungkan saat itu, seperti mengambil diamond terdekat dengan nilai tinggi, kembali ke base saat kapasitas hampir penuh, atau mengejar lawan yang membawa diamond. Bot juga mempertimbangkan penggunaan *teleporter* jika dapat mempersingkat jarak ke target, dan menekan tombol jika jumlah diamond di papan sudah sedikit.

Gerakan bot ditentukan berdasarkan arah tercepat menuju target, sambil menghindari rintangan seperti tembok dan bot lain. Jika arah terbaik terhalang, bot akan memilih alternatif yang valid, atau bergerak acak sebagai upaya terakhir.

1. **Implementasi Algoritma *Greedy* ke dalam Bot**

   Pada bot yang kami kembangkan (Ochobot), algoritma *greedy* diterapkan untuk memilih target terbaik secara lokal. Bot mengambil keputusan berdasarkan informasi yang tersedia saat itu saja, tanpa memperhitungkan langkah-langkah masa depan.

   Strategi greedy pada Ochobot dilakukan melalui:
   - Pemilihan target diamond berdasarkan kombinasi nilai (poin) dan jarak (menggunakan *manhattan distance*).
   - Pengambilan keputusan prioritas, seperti kembali ke base jika inventory penuh, atau mengejar bot lawan jika dekat dan membawa diamond.
   - Penggunaan fungsi *utilitas get_direction* untuk menentukan arah langkah optimal menuju goal.

Bot tidak menggunakan metode pencarian jalur (seperti BFS atau A*), sehingga semua keputusan bersifat lokal dan cepat, sesuai dengan prinsip *greedy.*

2. Cara Menjalankan Bot

Untuk menjalankan bot dalam permainan Diamonds, berikut langkah-langkah umumnya:

1. Menyusun kode logika bot dan menyimpannya dalam folder game.logic dengan nama file sesuai nama bot (contoh: ochobot.py).
2. Mendaftarkan bot ke server permainan dengan perintah run-bots.bat atau menjalankan perintah Python melalui main.py, misalnya:

   python main.py --logic Ochobot --email=your_email@example.com --name=Ochobot --password=your_password --team=timkita
3. Setelah berhasil mendaftar atau login, bot akan bergabung ke board permainan dan mulai bergerak otomatis.
4. Setiap giliran, bot akan menerima informasi *board,* menghitung langkah optimal, lalu mengirimkan perintah gerakan ke server.
5. Permainan akan terus berlangsung hingga waktu habis atau seluruh diamond habis.

Dengan demikian, seluruh proses berjalan secara otomatis dan *real-time,* serta keputusan bot ditentukan sepenuhnya oleh logika *greedy* yang telah diimplementasikan dalam kode program.

3. Penanganan Khusus

- *Teleport*: Jika jalur ke diamond terlalu jauh, bot mempertimbangkan apakah penggunaan *teleport* lebih efisien. Ini dihitung berdasarkan jarak sebelum dan sesudah teleportasi.
- Tombol (Diamond Button): Jika jumlah diamond yang tersisa di papan sangat sedikit (misal < 3), dan ada tombol di papan, bot akan menjadikannya tujuan untuk menambah diamond.
- Lawan: Jika ada lawan di sekitar yang membawa diamond, bot dapat beralih dari berburu diamond ke mengejar lawan tersebut, khususnya jika jaraknya cukup dekat (≤ 2 tile).

BAB III
APLIKASI STRATEGI GREEDY

3.1 Proses Mapping

Permainan Diamonds dapat diselesaikan dengan pendekatan algoritma *greedy* karena setiap langkah permainan merupakan pengambilan keputusan lokal yang dapat dievaluasi secara mandiri. Dalam konteks ini, strategi *greedy* digunakan untuk memilih langkah terbaik di setiap giliran berdasarkan kondisi lingkungan saat itu. Adapun pemetaan elemen-elemen *greedy* dalam permainan Diamonds adalah sebagai berikut:

1. Himpunan Kandidat (*Candidate Set*)

   Himpunan kandidat adalah kumpulan semua objek yang dapat dituju oleh bot dalam permainan. Objek-objek tersebut meliputi:
   - Diamond biru (bernilai 1 poin)
   - Diamond merah (bernilai 2 poin)
   - *Base* milik bot (tempat menyimpan diamond yang telah dikumpulkan)
   - *Red button* (untuk memunculkan kembali diamond jika sudah habis)
   - *Teleporter* (untuk berpindah lokasi dengan cepat)
   - Bot lawan yang sedang membawa diamond (dapat menjadi target *tackle*)

   Selain itu, langkah-langkah pergerakan yang mungkin dilakukan bot, yaitu ke arah atas, bawah, kiri, dan kanan, juga termasuk dalam kandidat pergerakan.

2. Himpunan Solusi (*Solution Set*)

   Himpunan solusi merupakan rangkaian keputusan pergerakan yang diambil bot selama permainan berlangsung. Setiap langkah akan menambah elemen pada himpunan solusi. Hasil akhir dari seluruh keputusan ini akan menentukan skor akhir bot.

3. Fungsi Solusi (*Solution Function*)

   Dalam konteks permainan Diamonds, fungsi solusi adalah fungsi yang menghitung seberapa baik solusi yang telah diambil. Karena permainan berakhir setelah waktu habis, maka fungsi solusi dinilai berdasarkan total skor yang berhasil dikumpulkan. Skor ini dihitung dari jumlah diamond yang dikembalikan ke base selama permainan.

4. Fungsi Seleksi (*Selection Function*)

   Fungsi seleksi digunakan untuk memilih langkah terbaik dari kumpulan kandidat yang tersedia. Pada bot Ochobot, fungsi seleksi didasarkan pada strategi *greedy* yang mempertimbangkan dua hal utama, yaitu jarak ke target (menggunakan *manhattan*

distance) dan nilai dari target tersebut (nilai diamond). Urutan pengambilan keputusan yang digunakan oleh bot adalah sebagai berikut:

- Jika jumlah diamond yang dibawa bot sudah mencapai 4, maka bot langsung memutuskan untuk kembali ke *base.*
- Jika terdapat bot lawan dalam jarak ≤ 2 yang sedang membawa diamond, maka bot akan mencoba melakukan *tackle*.
- Jika tidak ada kondisi khusus, maka bot akan memilih diamond terdekat yang memiliki nilai terbesar. Pemilihan ini dilakukan dengan mempertimbangkan nilai diamond (1 atau 2 poin) dan jarak dari bot ke diamond.
- Jika diamond sudah sedikit, dan *red button* tersedia, maka bot akan memprioritaskan menuju tombol.
- Jika tersedia *teleporter,* bot membandingkan apakah jalur melalui teleport bisa mempercepat perjalanan ke diamond.

Dengan strategi ini, bot selalu mengambil langkah terbaik secara lokal berdasarkan kondisi saat itu tanpa mempertimbangkan seluruh jalur ke depan. Hal ini sesuai dengan prinsip *greedy.*

5. Fungsi Kelayakan (*Feasibility Function*)

   Sebelum melakukan langkah, bot memeriksa apakah pergerakan tersebut valid dan aman. Kriteria kelayakan yang diperiksa adalah:
   - Apakah posisi tujuan berada dalam batas papan permainan.
   - Apakah posisi tersebut tidak ditempati oleh objek seperti dinding (*wall*) atau bot lain.
   - Jika bergerak ke arah tujuan akan menabrak atau menyebabkan *stuck*, maka bot memilih *fallback movement* (arah lain yang aman).

6. Fungsi Objektif (*Objective Function*)

   Tujuan utama dari strategi *greedy* ini adalah untuk memaksimalkan skor akhir. Skor diperoleh dari:
   - Diamond biru yang dikembalikan ke *base* → 1 poin per diamond
   - Diamond merah yang dikembalikan ke *base* → 2 poin per diamond

3.2 Eksplorasi Alternatif Solusi *Greedy*

Bot Ochobot menggunakan pendekatan *greedy* yang dikenal sebagai *greedy by density*. Strategi ini memilih target berdasarkan kombinasi antara nilai diamond dan jarak dari posisi bot. Artinya, bot akan lebih memilih diamond yang memberikan rasio nilai-per-jarak terbaik.

Secara rinci, strategi yang digunakan oleh bot dapat dijelaskan dalam beberapa mode berikut:

1. Mode *Collection*
   Merupakan mode default. Bot akan memilih diamond dengan perbandingan terbaik antara nilai dan jarak. Jika ada dua diamond dengan nilai sama, maka bot akan memilih yang paling dekat. Jika ada dua diamond dengan jarak sama, maka bot akan memilih yang nilainya lebih besar.
2. Mode *Return to Base*
   Aktif saat *inventory* diamond bot sudah mendekati penuh (≥ 4). Dalam mode ini, bot langsung menuju base untuk menyimpan diamond.
3. Mode *Tackle*
   Aktif jika terdapat bot lawan dalam jarak ≤ 2 langkah yang sedang membawa diamond. Bot akan mencoba menyerang untuk merebut diamond tersebut.
4. Mode *Red Button*
   Jika jumlah diamond di *board* sedikit (misalnya kurang dari 3) dan *red button* tersedia, maka bot akan menuju tombol untuk mereset diamond.
5. Mode *Teleport*
   Jika jalur menuju diamond lebih cepat menggunakan *teleporter* dibandingkan jalur langsung, maka bot akan memanfaatkan *teleport.*

3.3 Analisis Efisiensi dan Efektivitas Alternatif Solusi *Greedy*

Dalam permainan Diamonds, terdapat beberapa pendekatan strategi *greedy* yang dapat diterapkan untuk mengarahkan bot dalam mengambil keputusan setiap langkahnya. Masing-masing strategi memiliki tingkat efisiensi dan efektivitas yang berbeda, tergantung pada kondisi papan, jumlah pemain, serta distribusi objek seperti diamond dan lawan. Berikut adalah penjabaran beberapa alternatif strategi *greedy* beserta analisisnya:

Strategi pertama adalah *greedy* berdasarkan jarak terdekat. Strategi ini membuat bot selalu memilih diamond yang paling dekat, tanpa mempertimbangkan nilai diamond tersebut. Strategi ini sangat efisien karena proses seleksinya sederhana dan cepat, dengan kompleksitas waktu O(n), di mana n adalah jumlah diamond di papan. Namun dari segi efektivitas, strategi ini kurang optimal karena bot bisa saja mengabaikan diamond merah yang bernilai dua kali lebih besar jika posisinya hanya sedikit lebih jauh. Strategi ini cocok untuk permainan dengan banyak diamond tersebar merata dan tidak banyak kompetisi dari bot lain.

Strategi kedua adalah *greedy* berdasarkan nilai tertinggi. Dalam pendekatan ini, bot akan selalu memprioritaskan diamond dengan poin tertinggi terlebih dahulu, yaitu diamond merah. Strategi ini dapat meningkatkan perolehan skor jika diamond merah mudah diakses. Namun, secara efektivitas langkah, strategi ini bisa kurang efisien karena bot berisiko menempuh jarak jauh demi diamond bernilai tinggi, sementara diamond biru yang lebih dekat justru terlewatkan. Meskipun kompleksitas waktunya tetap O(n), strategi ini dapat menyebabkan pemborosan langkah jika tidak diimbangi dengan perhitungan jarak.

Strategi ketiga adalah *greedy by density*, yaitu strategi yang mempertimbangkan rasio nilai diamond terhadap jaraknya dari posisi bot (poin per langkah). Strategi ini berusaha mencari keseimbangan antara nilai yang diperoleh dan usaha yang diperlukan untuk mencapainya. Strategi ini sangat cocok untuk permainan seperti Diamonds yang memiliki batasan waktu dan langkah. Strategi *greedy by density* memiliki efisiensi yang tinggi (O(n)) dan memberikan efektivitas yang baik karena memperhitungkan *trade-off* antara nilai dan jarak. Strategi inilah yang digunakan oleh bot kami, Ochobot, karena mampu memberikan keputusan yang cepat, sederhana, namun tetap menguntungkan secara skor.

Selain ketiga strategi utama tersebut, ada pula strategi *oportunistik* seperti mode *tackle* terhadap bot lawan. Strategi ini efektif saat ada lawan dalam jangkauan pendek yang sedang membawa diamond. Dalam kondisi ini, mengejar lawan bisa menjadi cara cepat untuk memperoleh banyak poin. Namun, strategi ini juga membawa risiko, karena bisa saja bot tidak berhasil melakukan tackle atau justru ditabrak balik. Strategi ini lebih cocok dijadikan strategi tambahan, bukan strategi utama.

Terakhir, terdapat strategi yang memanfaatkan *red button*, yang dapat digunakan untuk memunculkan kembali diamond ketika jumlahnya sudah menipis. Strategi ini sangat berguna pada fase akhir permainan, terutama ketika area di sekitar base sudah tidak memiliki diamond lagi. Efisiensinya tinggi karena hanya melibatkan satu objek, namun efektivitasnya sangat tergantung pada waktu penggunaannya.

Dari berbagai alternatif yang telah dianalisis, strategi *greedy by density* dipilih sebagai strategi utama dalam pengembangan bot Ochobot. Strategi ini menawarkan kombinasi terbaik antara efisiensi pengambilan keputusan dan efektivitas skor yang diperoleh. Strategi ini juga *fleksibel* dan *adaptif* terhadap perubahan kondisi permainan. Dengan mempertimbangkan nilai dan jarak secara bersamaan, bot dapat mengambil keputusan yang mengarah pada hasil optimal tanpa perlu menyimpan histori atau melakukan pencarian jalur yang kompleks.

3.4 Strategi *Greedy* yang Dipilih

Strategi *greedy* yang diterapkan pada bot Ochobot dipilih karena:

- Sederhana dan efisien. Bot tidak melakukan pencarian jalur yang kompleks seperti BFS atau A*, sehingga keputusan dapat diambil dengan cepat.
- Adaptif terhadap berbagai kondisi permainan. Bot akan mengubah tujuannya tergantung pada keadaan di sekitarnya.
- Relevan dengan kebutuhan permainan. Dalam permainan ini, kecepatan mengambil keputusan dan *fleksibilitas* terhadap kondisi sangat penting.
- Konsisten dengan prinsip *greedy.* Bot selalu mengambil keputusan terbaik saat ini tanpa mempertimbangkan langkah-langkah masa depan, yang merupakan inti dari algoritma *greedy.*

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *Greedy*

1. Pseudocode

```
import random # import random untuk memilih arah secara acak
from typing import Optional # List, Tuple untuk tipe data yang
digunakan

from game.logic.base import BaseLogic # adalah kelas dasar untuk
logika permainan
from game.models import GameObject, Board, Position # adalah model
yang digunakan dalam permainan
from ..util import get_direction # adalah fungsi utilitas untuk
mendapatkan arah dari satu posisi ke posisi lain

# ini adalah kelas Ochobot yang merupakan turunan dari BaseLogic
class Ochobot(BaseLogic):
    def __init__(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)] #
arah gerakan: kanan, bawah, kiri, atas
        self.goal_position: Optional[Position] = None
        self.current_direction = 0
        self.board_width = 15
        self.board_height = 15

    # fungsi manhattan_distance untuk menghitung jarak Manhattan
antara dua posisi
    def manhattan_distance(self, pos1: Position, pos2: tuple):
        return abs(pos1.x - pos2[0]) + abs(pos1.y - pos2[1])

    # fungsi untuk memeriksa apakah posisi yang diberikan valid
dalam batas papan
    def is_valid_position(self, x: int, y: int) -> bool:
        return 0 <= x < self.board_width and 0 <= y <
self.board_height

    # fungsi untuk mendapatkan arah dari posisi saat ini ke posisi
```

```
tujuan
    def next_move(self, board_bot: GameObject, board: Board):
        props = board_bot.properties
        current_position = board_bot.position
        my_x, my_y = current_position.x, current_position.y

        # === IMPLEMENTASI STRATEGI GREEDY ===
        # GREEDY DECISION 1: Jika diamond sudah 4 atau lebih,
pulang ke base (lokal optimal: amankan poin sekarang)
        if props.diamonds >= 4:
            self.goal_position = props.base
        else:
            # Ambil semua diamond di papan
            diamonds = [
                obj for obj in board.game_objects
                if obj.type == "DiamondGameObject"
            ]
            # Ambil teleporter di papan
            teleporters = [
                obj for obj in board.game_objects
                if obj.type == "TeleporterGameObject"
            ]
            # Ambil red button di papan
            button = next(
                (obj for obj in board.game_objects if obj.type ==
"DiamondButtonGameObject"),
                None
            )
            # tackle lawan terdekat yang memiliki diamond
            opponents = [
                p for p in board.bots
                if p != board_bot and p.properties.diamonds > 0
            ]

            # GREEDY DECISION 2: Jika ada musuh dengan diamond di
radius <= 2, dekati untuk tackle (high value gain)
            # menentukan posisi tujuan berdasarkan kondisi
permainan
            if opponents:
```

```
                nearest_opponent = min(
                    opponents,
                    key=lambda p:
self.manhattan_distance(current_position, (p.position.x,
p.position.y))
                )
                # jika jarak ke lawan terdekat kurang dari atau
sama dengan 2, set tujuan ke posisi lawan
                if self.manhattan_distance(current_position,
(nearest_opponent.position.x, nearest_opponent.position.y)) <= 2:
                    self.goal_position = nearest_opponent.position
                else:
                    self.goal_position = None
            else:
                self.goal_position = None

            # GREEDY DECISION 3: Jika tidak ada lawan dekat, cari
diamond terdekat & bernilai tinggi
            if not self.goal_position and diamonds:
                target_diamond = min(
                    diamonds,
                    key=lambda d: (
                        self.manhattan_distance(current_position,
(d.position.x, d.position.y)),
                        -d.properties.points
                    )
                )
                self.goal_position = target_diamond.position

            # GREEDY DECISION 4: Jika tidak ada diamond terdekat,
cek teleportasi
            if not self.goal_position and teleporters and
len(teleporters) == 2:
                t1, t2 = teleporters
                t1_pos = (t1.position.x, t1.position.y)
                t2_pos = (t2.position.x, t2.position.y)
                # jika ada diamond, tentukan tujuan berdasarkan
jarak ke diamond setelah teleportasi
                if diamonds:
```

```
                    nearest_diamond = min(
                        diamonds,
                        key=lambda d:
self.manhattan_distance(current_position, (d.position.x,
d.position.y))
                    )
                    diamond_pos = (nearest_diamond.position.x,
nearest_diamond.position.y)
                    dist_to_diamond =
self.manhattan_distance(current_position, diamond_pos)
                    dist_t1 =
self.manhattan_distance(current_position, t1_pos)
                    dist_t2 =
self.manhattan_distance(current_position, t2_pos)
                    dist_after_t1 =
self.manhattan_distance(Position(t2_pos[0], t2_pos[1]),
diamond_pos)
                    dist_after_t2 =
self.manhattan_distance(Position(t1_pos[0], t1_pos[1]),
diamond_pos)
                    if dist_t1 + dist_after_t1 < dist_to_diamond:
                        self.goal_position = t1.position
                    elif dist_t2 + dist_after_t2 <
dist_to_diamond:
                        self.goal_position = t2.position

            # GREEDY DECISION 5: Jika tidak ada diamond terdekat,
cek red button
            if not self.goal_position and button and len(diamonds)
< 3:
                self.goal_position = button.position

            # GREEDY DECISION 6: Jika tidak ada tujuan, pulang ke
base
            if not self.goal_position:
                self.goal_position = props.base

        #  # === IMPLEMENTASI GERAKAN MENUJU TARGET ===
        # jika ada tujuan, hitung arah gerakan menuju tujuan
```

```
        if self.goal_position:
            delta_x, delta_y = get_direction(
                current_position.x,
                current_position.y,
                self.goal_position.x,
                self.goal_position.y,
            )
            next_x, next_y = my_x + delta_x, my_y + delta_y

            if not self.is_valid_position(next_x, next_y):
                delta_x, delta_y = 0, 0
            else:
                obstacles = [
                    (obj.position.x, obj.position.y)
                    for obj in board.game_objects
                    if obj.type in ["WallGameObject",
"BotGameObject"]
                ]
                if (next_x, next_y) in obstacles and not
(self.goal_position.x == next_x and self.goal_position.y ==
next_y):
                    delta_x, delta_y = 0, 0

        # === JIKA TERJEBAK, CARI ARAH LAIN SECARA GREEDY ===
        # jika tidak ada tujuan atau arah gerakan tidak valid,
pilih arah acak
        if delta_x == 0 and delta_y == 0:
            valid_moves = [
                (dx, dy) for dx, dy in self.directions
                if self.is_valid_position(my_x + dx, my_y + dy)
                and (my_x + dx, my_y + dy) not in [
                    (obj.position.x, obj.position.y)
                    for obj in board.game_objects
                    if obj.type in ["WallGameObject",
"BotGameObject"]
                ]
            ]
            # jika ada gerakan yang valid, pilih salah satu secara
acak
```

```
            if valid_moves:
                delta_x, delta_y = random.choice(valid_moves)
            else:
                # Jika semua arah terhalang, pilih arah acak
                delta_x, delta_y =
self.directions[self.current_direction]
                self.current_direction = (self.current_direction +
1) % len(self.directions)

        # jika arah gerakan valid, kembalikan delta_x dan delta_y
        return delta_x, delta_y
```

2. **Penjelasan Alur Program**

Program bot Ochobot dirancang untuk mengambil keputusan secara otomatis berdasarkan strategi *greedy* yang telah diimplementasikan. Alur kerja program ini dijelaskan sebagai berikut:

1. Inisialisasi Bot

Saat objek bot Ochobot diinisialisasi, bot menetapkan empat arah gerakan utama yaitu kanan, bawah, kiri, dan atas. Bot juga menyimpan posisi tujuan (*goal_position*) yang awalnya kosong, serta mengatur ukuran papan permainan menjadi 15x15. Variabel *current_direction* digunakan untuk menangani gerakan saat bot berada dalam posisi terjebak.

2. Perhitungan Jarak Manhattan

Bot menggunakan fungsi *manhattan_distance* untuk menghitung jarak antara dua titik di papan permainan. Perhitungan ini penting untuk mengevaluasi posisi target seperti diamond, musuh, *base,* dan lainnya. Jarak ini dihitung dengan menjumlahkan selisih *absolut* koordinat x dan y dari dua posisi.

3. Validasi Posisi

Sebelum bergerak ke posisi tertentu, bot memeriksa validitas posisi tersebut melalui fungsi *is_valid_position*, yaitu memastikan bahwa koordinat tujuan tidak berada di luar batas papan permainan (0 hingga 14 untuk x dan y).

4. Pengambilan Keputusan Utama (next_move)

Fungsi utama *next_move* dijalankan setiap giliran untuk menentukan langkah bot. Di dalamnya, bot melakukan pengambilan keputusan berdasarkan kondisi permainan sebagai berikut:

- Pemeriksaan *Inventory*: Jika bot telah membawa minimal 4 diamond, maka secara otomatis akan kembali ke base untuk mengamankan poin.

- Identifikasi Objek Penting: Bot mengumpulkan data dari papan permainan, seperti semua diamond, *teleporter, red button*, dan bot lawan yang membawa diamond.
- Prioritas *Tackle*: Jika terdapat musuh dalam jarak dua langkah dan membawa diamond, maka bot akan mendekati musuh tersebut untuk mencoba melakukan *tackle.*
- Pemilihan Diamond: Jika tidak ada lawan yang layak di*tackle,* maka bot mencari diamond terdekat dengan nilai tertinggi. Pemilihan dilakukan berdasarkan kombinasi jarak dan poin (strategi *greedy by density*).
- Pemanfaatan *Teleporter*: Jika tersedia dua *teleporter*, bot menghitung apakah menggunakan *teleporter* dapat mempercepat pergerakan ke diamond dibandingkan jalur biasa. Jika ya, maka *teleporter* dijadikan tujuan.
- Aktivasi *Red Button*: Jika jumlah diamond di papan sangat sedikit dan red button tersedia, maka bot akan menuju tombol tersebut untuk memunculkan kembali diamond.
- *Fallback* ke *Base*: Jika tidak ada target lain yang layak, maka bot akan kembali ke base.

5. Eksekusi Gerakan

Jika tujuan (*goal_position*) telah ditentukan, maka bot akan menghitung arah pergerakan menggunakan fungsi *get_direction,* dan mencoba bergerak menuju titik tersebut. Jika arah yang dituju tidak valid atau terhalang oleh objek seperti tembok atau bot lain, maka gerakan dibatalkan.

6. Penanganan Saat Terjebak

Jika arah yang dituju tidak valid dan tidak ada gerakan lain yang layak, bot akan mencari arah lain secara acak dari arah yang masih memungkinkan. Jika semua arah terblokir, bot akan bergantian memilih arah berdasarkan rotasi untuk keluar dari posisi terkunci.

7. Pengembalian Arah Gerakan

Terakhir, bot akan mengembalikan pasangan nilai (delta_x, delta_y) sebagai instruksi arah gerakan untuk giliran tersebut. Nilai ini akan dieksekusi oleh *engine* untuk menggerakkan bot di papan permainan.

Dengan alur kerja ini, bot Ochobot mampu beradaptasi terhadap berbagai kondisi permainan dan membuat keputusan secara cepat tanpa perencanaan jalur kompleks, sesuai dengan prinsip strategi *greedy* yang menjadi dasar logikanya.

4.2 Struktur Data yang Digunakan

Dalam pengembangan bot Ochobot, terdapat beberapa struktur data utama yang digunakan secara langsung dalam implementasi algoritma *greedy.* Struktur data ini berasal dari pustaka game Diamonds dan sangat penting untuk pengambilan keputusan pada setiap langkah permainan. Berikut adalah struktur data yang digunakan:

1. Objek *GameObject*
   GameObject adalah representasi umum dari seluruh objek dalam permainan, seperti diamond, bot musuh, *teleportasi*, dan *red button*. Setiap objek ini memiliki atribut penting seperti *position* (berisi koordinat x dan y), type (misalnya: "DiamondGameObject" atau "TeleporterGameObject"), dan *properties.* Dalam konteks diamond, properties ini mencakup nilai poin dari diamond tersebut (poin). Sedangkan pada bot, properties berisi informasi seperti jumlah diamond yang dibawa, posisi *base*, dan kapasitas *inventory*. Bot juga menggunakan informasi ini untuk mengevaluasi apakah perlu kembali ke *base.*

2. Objek *Board*
   Board merupakan representasi keseluruhan papan permainan yang mencakup dimensi papan (*width* dan *height*), daftar semua *GameObject* yang sedang berada di papan (game_objects), serta daftar seluruh bot. Struktur ini digunakan untuk mengetahui kondisi sekitar bot, seperti posisi lawan, posisi diamond, dan kemungkinan rintangan (seperti dinding atau bot lain). Informasi dari objek *Board* menjadi dasar utama pengambilan keputusan *greedy.*

3. Objek *Position*
   Position adalah struktur data yang menyimpan koordinat lokasi x dan y suatu objek di papan. *Position* digunakan dalam fungsi evaluasi seperti *manhattan_distance*, untuk menghitung jarak antara posisi bot dan objek target (misal: diamond, *base*, atau lawan). Bot menggunakan hasil perhitungan ini untuk memilih objek dengan jarak terdekat atau *density* terbaik.

4. Properti dari *props* pada Bot Sendiri
   Setiap bot memiliki properti individual yang disimpan dalam *props*. Ini termasuk:
   - diamonds: jumlah diamond yang sedang dibawa
   - *inventory_size:* kapasitas maksimum diamond yang dapat dibawa
   - *base*: posisi *base* bot
   - *milliseconds_left:* sisa waktu permainan dalam milidetik
     Properti ini sangat penting karena menentukan kapan bot harus berhenti mengumpulkan diamond dan kembali ke *base,* serta memastikan bot tidak kehilangan poin karena *inventory* penuh atau waktu habis.

5. List dan Filter Dinamis
   Dalam algoritma bot, struktur data berbasis list juga digunakan untuk menyaring dan menyortir objek yang relevan. Contohnya:
   - List diamonds disaring berdasarkan type == "DiamondGameObject"
   - List *teleporters* disaring untuk mencari jalur tercepat

- List *opponents* digunakan untuk mengevaluasi potensi *tackle*.
- *Filtering* ini membantu menyusun kandidat solusi sebelum dilakukan evaluasi *greedy* berdasarkan jarak, poin, dan kondisi permainan.

Secara keseluruhan, struktur data yang digunakan bersifat modular dan dinamis, memungkinkan bot untuk mengevaluasi kondisi papan secara efisien dan menentukan langkah terbaik secara lokal. Kombinasi data spasial (seperti posisi) dan properti objek (seperti poin diamond dan *inventory*) mendukung strategi *greedy* dalam menentukan target yang optimal di setiap giliran.

4.3 Analisis Desain Solusi Algoritma

Solusi algoritma greedy yang diterapkan pada bot Ochobot dirancang untuk memilih aksi terbaik secara lokal berdasarkan prioritas kondisi permainan. Strategi *greedy* yang digunakan adalah memilih objek tujuan (*goal*) berdasarkan evaluasi poin, jarak, serta kondisi sekitar. Berikut ini adalah langkah-langkah evaluasi dan strategi pemilihan keputusan oleh bot:

a. Jarak bot terhadap setiap objek penting
   Bot menghitung jarak *Manhattan* terhadap diamond, musuh terdekat, *base, teleporter,* dan *red button* untuk menentukan target terdekat atau ancaman terdekat.

b. Prioritas diamond berdasarkan *density*
   Jika *inventory* belum penuh, bot akan memilih diamond dengan *density* terbaik, yaitu poin dibagi jarak. Jika terdapat beberapa diamond dengan *density* serupa, maka diamond dengan jarak lebih dekat dipilih.

c. Evaluasi kondisi musuh di sekitar
   Jika terdapat musuh yang membawa diamond dan berada dalam radius 2 langkah, bot akan mempertimbangkan untuk mendekat dan melakukan *tackle*. Ini menjadi bentuk *greedy* terhadap potensi rampasan poin.

d. Penggunaan *teleporter* sebagai alternatif jalur
   Jika terdapat dua *teleporter* dan rute melalui *teleporter* mengurangi jarak ke diamond dibanding jalur langsung, maka bot akan diarahkan ke *teleporter* yang relevan.

e. Prioritas ke *red button* ketika diamond langka
   Jika jumlah diamond di papan sedikit (misal: < 3), dan tidak ada target yang layak diambil, maka bot diarahkan ke *red button* untuk memicu regenerasi diamond.

f. Kondisi kembali ke *base*
   Bot akan kembali ke *base* dalam tiga kondisi utama: (1) diamond yang dibawa sudah

maksimal, (2) langkah ke *base* sama dengan waktu tersisa, atau (3) diamond yang dibawa = 4 dan target berikutnya bernilai 2 poin.

Strategi ini secara umum efektif untuk kasus permainan dengan distribusi diamond yang seimbang. Namun, seperti kebanyakan algoritma *greedy*, strategi ini tidak mempertimbangkan konsekuensi jangka panjang. Dalam situasi di mana musuh lebih cepat ke target atau posisi bot terjepit, solusi ini bisa gagal menghasilkan nilai optimal. Meski begitu, pendekatan *greedy* yang digunakan memberikan performa baik dalam banyak skenario umum karena waktu eksekusinya efisien dan pengambilan keputusannya cukup adaptif terhadap kondisi permainan.

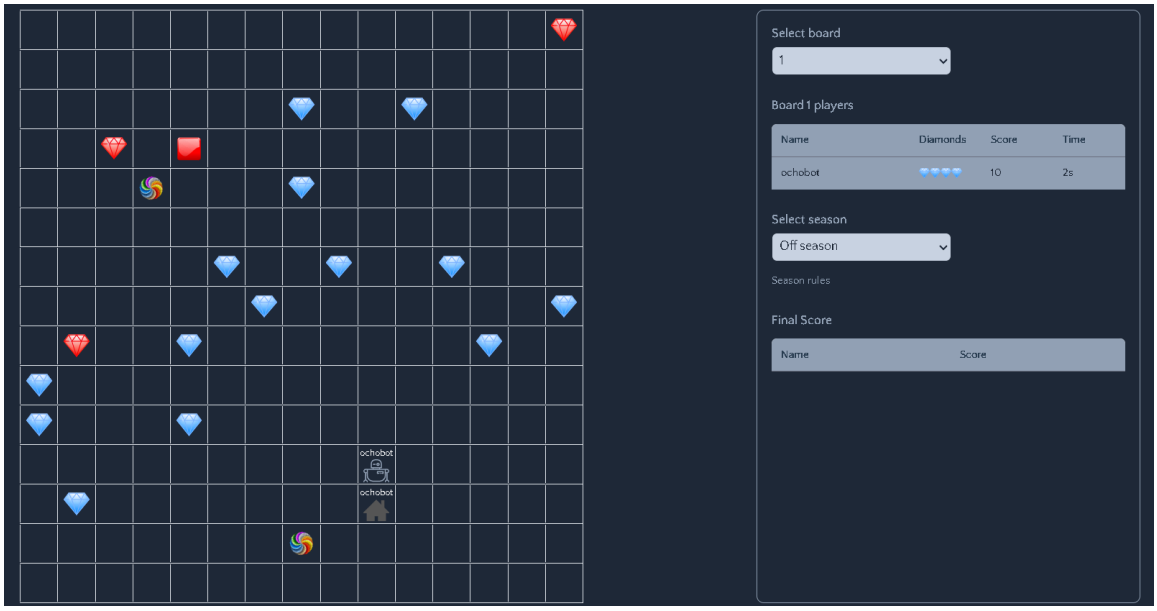
4.4 Pengujan Bot

1. **Skenario pengujian**
   Untuk memastikan bahwa strategi *greedy* yang diimplementasikan pada bot Ochobot berjalan sesuai harapan, dilakukan serangkaian skenario pengujian berdasarkan kondisi permainan yang merepresentasikan kemungkinan-kemungkinan nyata yang muncul. Pengujian ini bertujuan untuk mengevaluasi apakah keputusan bot sesuai dengan prioritas *greedy* yang telah dirancang. Berikut adalah skenario-skenario yang diuji:

Skenario	Tujuan
Bot memiliki 4 diamond, berada 5 langkah dari *base*	Menguji apakah bot akan langsung kembali ke *base*
Musuh dengan diamond berada dalam jarak 2 langkah	Menguji apakah bot akan mengejar musuh untuk melakukan *tackle*
Dua diamond tersedia: satu dekat bernilai 1, satu jauh bernilai 2	Menguji apakah bot memilih diamond berdasarkan rasio poin per jarak
Diamond sangat sedikit dan *red button* berada dekat	Menguji apakah bot memprioritaskan *red button* untuk regenerasi diamond
Diamond jauh tapi bisa dijangkau lebih cepat dengan *teleportasi*	Menguji apakah bot memilih *teleport* untuk efisiensi r*ute*

2. **Hasil Pengujian dan Analisis**

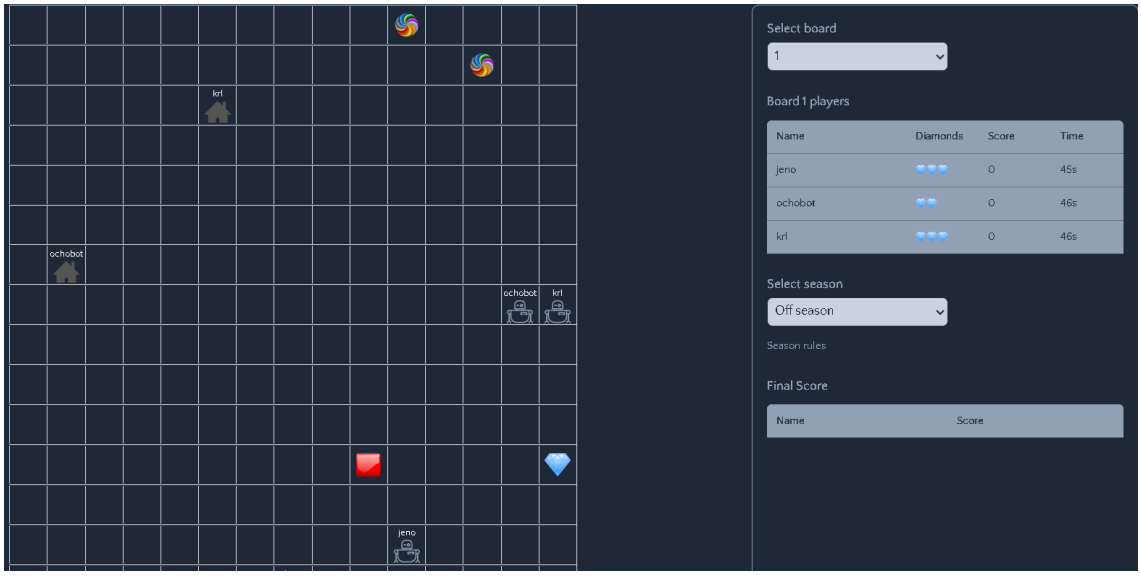
   Skenario 1 – Bot kembali ke *base* saat membawa 4 diamond
   Pada pengujian ini, bot dibiarkan mengambil diamond hingga membawa 4 poin. Terlihat bahwa setelah jumlah diamond mencukupi, bot secara otomatis langsung menuju base tanpa mengejar diamond lain di sekitarnya. Ini menunjukkan bahwa strategi *greedy* untuk mengamankan poin lokal telah dijalankan dengan benar.



Karena sudah memiliki 4 diamond, bot lebih prefer kembali ke base di banding menghabiskan waktu untuk mengambil diamond yang jauh

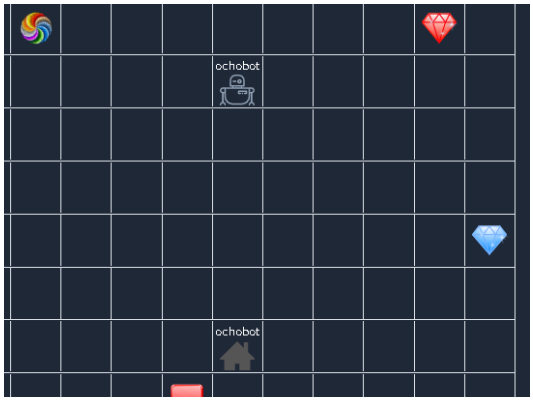
Skenario 2 – Bot mengejar dan tackle musuh dalam jarak ≤ 2 langkah

Dalam skenario ini, bot musuh yang membawa diamond sengaja diletakkan dekat dengan bot utama. Hasilnya, bot langsung berpindah arah menuju musuh dan melakukan tackle yang berhasil. Perilaku ini menunjukkan bahwa fungsi *tackle* bot aktif dan efektif dalam menambah poin secara cepat dari musuh.



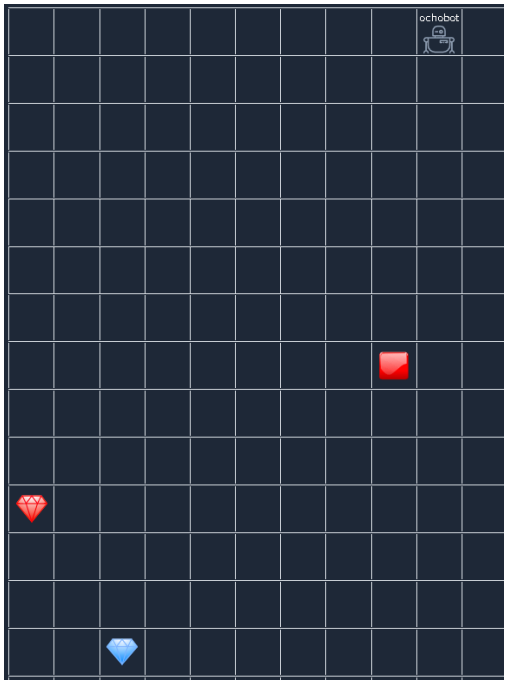
Disini Ochobot akan men-tackle bot krl karena krl sedang memiliki 3 diamond dan step nya kurang dari atau sama dengan 2, maka dari itu ochobot akan berusaha men-tackle krl

Skenario 3 – Pemilihan diamond berdasarkan rasio nilai per jarak (density)
Dalam pengujian ini, terdapat dua diamond: satu biru yang dekat, dan satu merah yang lebih jauh. Bot memilih menuju diamond merah meskipun lebih jauh, karena nilai 2 poin yang ditawarkan lebih tinggi dibandingkan rasio jarak ke diamond biru. Ini menunjukkan implementasi *greedy by density* berjalan dengan baik.



Disini bot yang bermula dari base langsung menuju diamond merah meskipun jarak ke diamond biru lebih dekat

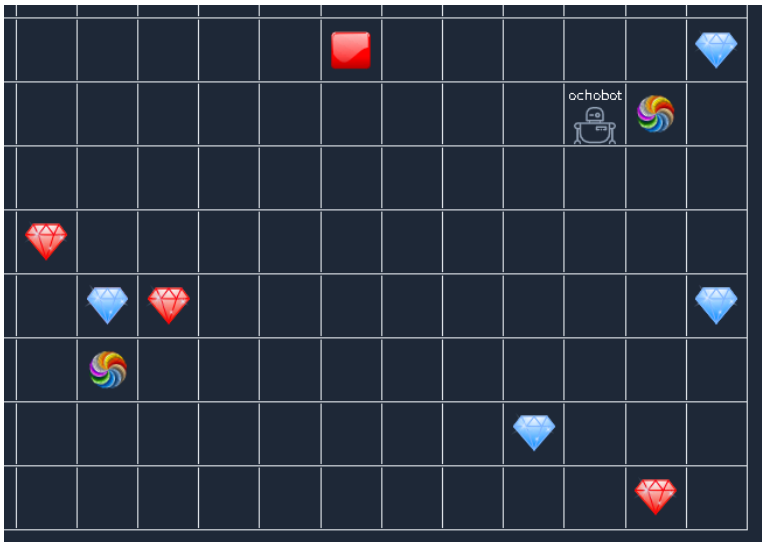
Skenario 4 – Bot menekan red button saat jumlah diamond sedikit
Ketika jumlah diamond di papan tinggal dua dan *red button* berada dalam jangkauan, bot segera berpindah ke *red button* dan menekannya. Hal ini menunjukkan bahwa kondisi khusus untuk memunculkan kembali diamond telah dijalankan secara otomatis.



Disini terdapat kondisi dimana bot akan menekan red button karena sisa diamond hanya ada 2

Skenario 5 – Bot menggunakan teleportasi untuk efisiensi jalur

Bot ditempatkan dalam situasi di mana jalur langsung ke diamond lebih panjang dibandingkan jika menggunakan *teleporter.* Bot memilih salah satu *teleporter* dan berhasil memperpendek jarak ke diamond. Ini membuktikan bahwa perbandingan jalur *teleport* vs langsung telah dihitung dengan baik oleh algoritma.



Bot akan memilih teleporter karena diamond di dekat teleporter lainnya lebih menguntungka

BAB V
KESIMPULAN DAN SARAN

5. 1 Kesimpulan

Implementasi algoritma *greedy* pada bot Ochobot dalam permainan Diamonds terbukti berhasil dan efektif. Bot mampu mengambil keputusan optimal secara lokal tanpa pencarian jalur kompleks, dengan elemen *greedy* yang terdefinisi jelas. Strategi *greedy by density* memberikan hasil skor yang maksimal dengan efisiensi waktu yang baik, serta menunjukkan kemampuan adaptasi terhadap berbagai kondisi permainan seperti *inventory* penuh, ancaman lawan, dan penggunaan *teleporter*. Algoritma ini juga efisien secara komputasi dengan kompleksitas O(n) per langkah. Bot Ochobot menunjukkan bahwa algoritma *greedy*, meskipun sederhana, dapat sangat adaptif dan efektif dalam lingkungan permainan dinamis seperti Diamonds, memungkinkan pengambilan keputusan yang cepat tanpa memerlukan pencarian jalur yang kompleks.

5.2 Saran

Sebaiknya ada optimasi lebih lanjut pada fungsi seleksi, agar bot dapat mempertimbangkan risiko dari bot lawan dan kondisi lain yang lebih kompleks. Penanganan situasi terjebak juga perlu ditingkatkan untuk keputusan yang lebih cerdas, agar dapat mengidentifikasi kelemahan strategi dan menemukan peluang perbaikan. Diharapkan bot Diamonds dapat terus berkembang dan memberikan performa optimal.

LAMPIRAN

A. R*epository* GitHub:
https://github.com/muhammad-nurikhsan/Tubes1_ELEVENTWELFTH

B. Video Penjelasan:
https://drive.google.com/drive/folders/1FaBcXYScwzzYBIBXKRNC7MtMwC6EYI-4?usp=sharing

DAFTAR PUSTAKA

1. Munir, Rinaldi. “Algoritma Greedy (Bagian 1)”. (https://kuliah.itera.ac.id/mod/resource/view.php?id=32758)
2. Munir, Rinaldi. “Algoritma Greedy (Bagian 2 & 3)”. (https://kuliah.itera.ac.id/mod/resource/view.php?id=33150)
3. FIKTI UMSU. 2025. “Algoritma Greedy: Pengertian, Jenis dan Contoh Program.” https://fikti.umsu.ac.id/algoritma-greedy-pengertian-jenis-dan-contoh-program/.
4. Wijayanti, Rini, Wahyu Nugraha, and Kusrini. 2021. “Optimalisasi Penyelesaian Permainan pada Game Puzzle 8 dengan Perbandingan Algoritma A* dan Greedy.” https://citec.amikom.ac.id/main/index.php/citec/article/view/230/167.