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 4 (BotNation)

Stevanus Cahya Anggara	123140038
Muhammad Romadhon Santoso	123140031
Annisa Al-Qoriah	123140030

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

Tugas Besar IF2211 Strategi Algoritma ini bertujuan untuk mengimplementasikan algoritma greedy dalam pembuatan bot yang akan digunakan dalam permainan Diamonds. Permainan ini merupakan sebuah tantangan pemrograman yang mengharuskan para peserta untuk membuat bot yang dapat bersaing dengan bot dari peserta lainnya. Setiap bot bertujuan untuk mengumpulkan diamond sebanyak-banyaknya dengan cara yang efisien. Bot yang berhasil mengumpulkan diamond terbanyak akan memenangkan permainan.

Deskripsi Permainan Diamonds

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 Diamonds

Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi greedy dalam membuat bot ini. Program permainan Diamonds terdiri atas 1) Game engine, yang secara umum berisi:

A. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot

B. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan

2) Bot starter pack, yang secara umum berisi:

A. Program untuk memanggil API yang tersedia pada backend
B. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)
C. Program utama (main) dan utilitas lainnya

Terdapat pula komponen-komponen dari permainan Diamonds antara lain, yakni sebagai berikut

1) Diamonds

Gambar 2. Diamond biru dan merah

Untuk memenangkan pertandingan ini, 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

Gambar 3. Red 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) Teleporter

Gambar 4. Teleportes

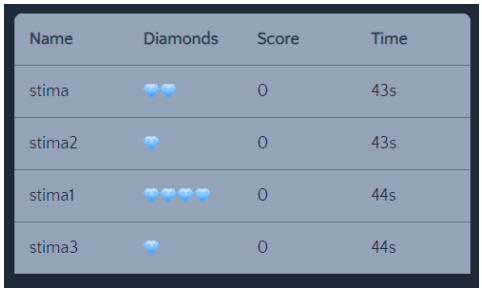
Terdapat 2 *teleporter* yang saling terhubung satu sama lain. Jika bot melewati sebuah *teleporter* maka bot akan berpindah menuju posisi *teleporter* yang lain.

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 di bawah) bot menjadi kosong.

5) Inventory



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.

Tujuan Tugas Besar

Tujuan dari tugas besar ini adalah untuk mengimplementasikan sebuah bot yang menggunakan algoritma greedy untuk memenangkan permainan Diamonds. Bot yang dikembangkan akan diuji dalam kompetisi dengan bot-bot dari kelompok lain. Dalam hal ini, setiap bot harus menggunakan strategi greedy yang sesuai dengan objektif permainan, yakni mengumpulkan diamond sebanyak mungkin dan memaksimalkan skor, serta memastikan diamond yang telah dikumpulkan tidak dicuri oleh bot lawan.

Tugas ini mengharuskan mahasiswa untuk mengeksplorasi beberapa alternatif strategi greedy yang dapat digunakan dalam permainan ini, mengembangkan solusi berdasarkan algoritma greedy yang efisien, serta mengimplementasikan solusi tersebut ke dalam bot. Dengan demikian, mahasiswa tidak hanya belajar mengembangkan algoritma, tetapi juga mengasah keterampilan dalam menciptakan sistem yang berkompetisi dalam lingkungan yang dinamis.

Ruang Lingkup Tugas

Tugas besar ini mencakup pengembangan bot permainan yang menggunakan algoritma greedy untuk mengumpulkan diamond dalam permainan Diamonds. Setiap kelompok diberikan game engine dan bot starter pack yang telah disediakan. Game engine ini sudah mencakup backend dan frontend permainan, serta API untuk komunikasi antara bot dan game engine. Sedangkan Bot Starter Pack berisi program untuk memanggil API yang tersedia pada backend, Program bot logic, dan Program utama (main) dan utilitas lainnya. Mahasiswa diminta untuk mengembangkan logika permainan bot menggunakan algoritma greedy, yang mencakup berbagai keputusan strategis dalam pengambilan langkah dan pengumpulan diamond.

Beberapa alternatif strategi greedy yang dipertimbangkan antara lain mengutamakan pengambilan diamond dengan nilai tertinggi, memilih langkah berdasarkan jarak terdekat ke diamond, memanfaatkan teleporter untuk mempercepat perjalanan, atau memilih posisi yang memiliki rerata jarak terdekat ke banyak diamond. Mahasiswa juga akan diminta untuk mengeksplorasi berbagai kemungkinan strategi dan menganalisis efisiensi serta efektivitasnya dalam permainan.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy adalah sebuah pendekatan dalam penyelesaian masalah optimasi yang berfokus pada pengambilan keputusan lokal terbaik pada setiap langkahnya, dengan tujuan untuk menghasilkan solusi global yang optimal. Prinsip dasar algoritma ini adalah untuk memilih solusi terbaik yang tersedia pada tiap langkah, tanpa mempertimbangkan pilihan-pilihan sebelumnya atau potensi solusi masa depan. Dalam istilah sederhana, algoritma greedy berusaha membuat keputusan yang terbaik pada saat itu juga, dengan harapan bahwa keputusan-keputusan tersebut akan mengarah pada solusi optimal secara keseluruhan. Meskipun cara ini cukup efisien dan sederhana untuk diterapkan, penting untuk dicatat bahwa algoritma greedy tidak selalu menghasilkan solusi optimal global untuk semua jenis masalah [1].

Proses kerja algoritma greedy dimulai dengan memilih langkah pertama yang tampaknya terbaik. Setelah itu, langkah-langkah berikutnya dipilih secara berulang-ulang, dengan tetap mempertahankan prinsip pemilihan lokal terbaik. Dalam beberapa kasus, pendekatan ini berhasil menghasilkan solusi optimal global, tetapi dalam masalah lain, solusi yang ditemukan mungkin hanya lokal optimal, tidak global optimal. Ini karena algoritma greedy tidak memandang keseluruhan ruang solusi, melainkan hanya memfokuskan pada keputusan terbaik pada setiap titik waktu.

Pada dasarnya, algoritma greedy diterapkan pada berbagai masalah optimasi, antara lain masalah pencocokan, pemrograman linier, penjadwalan, dan lain-lain. Beberapa masalah yang terkenal dan sering diselesaikan menggunakan algoritma greedy adalah masalah knapsack, masalah minimum spanning tree (MST) seperti yang ditemukan pada algoritma Prim dan Kruskal, serta masalah Huffman encoding yang digunakan dalam kompresi data. Dalam masalah knapsack, misalnya, algoritma ini memilih barang yang akan dimasukkan ke dalam tas berdasarkan rasio nilai terhadap berat barang. Meskipun algoritma greedy dapat memberikan solusi yang sangat efisien dalam hal waktu komputasi, solusinya tidak selalu optimal pada semua kasus.

Masalah minimum spanning tree, seperti yang ditangani oleh algoritma Prim atau Kruskal, juga merupakan contoh aplikasi algoritma greedy di mana tujuan utamanya adalah untuk menghubungkan semua titik dalam graf dengan total bobot minimum. Sementara itu, pada Huffman encoding, algoritma ini digunakan untuk menghasilkan kode biner yang paling efisien untuk pengkodean simbol berdasarkan frekuensinya, yang merupakan bagian penting dalam kompresi data.

Meskipun algoritma greedy sangat efektif dalam beberapa kasus, ia memiliki kelemahan utama, yaitu ketidakmampuannya untuk menjamin solusi global optimal dalam setiap situasi. Algoritma ini cenderung bekerja baik hanya pada masalah-masalah yang memenuhi kondisi tertentu, yang sering disebut sebagai greedy choice property dan optimal substructure. Greedy choice property memastikan bahwa memilih pilihan terbaik pada setiap langkah lokal akan memberikan solusi terbaik global, sementara optimal substructure mengindikasikan bahwa solusi optimal untuk masalah tersebut dapat dibangun secara bertahap dari solusi optimal sub-masalahnya.

Secara keseluruhan, meskipun algoritma greedy menawarkan pendekatan yang mudah dan efisien dalam banyak situasi, pemilihan algoritma yang tepat tetap harus didasarkan pada analisis mendalam terhadap struktur masalah yang dihadapi.

2.2 Cara Kerja Program

Program ini adalah implementasi dari bot permainan (game bot) yang menggunakan strategi greedy untuk mengumpulkan diamond (diamond) dengan cara yang efisien. Bot ini didesain untuk memaksimalkan jumlah diamond yang dikumpulkan dalam waktu terbatas dan memastikan bahwa ia dapat kembali ke markas (base) tepat waktu, tanpa kehilangan kesempatan untuk mengumpulkan diamond sebanyak mungkin.

2.2.1 Cara Implementasi Program

Cara kerja program ini dimulai dengan pemindaian peta permainan (board) yang dilakukan pada setiap giliran untuk mengidentifikasi posisi diamond yang tersedia. Dengan menggunakan informasi tersebut, bot kemudian harus membuat keputusan berdasarkan nilai lokal yang paling menguntungkan. Prinsip dasar dari strategi greedy adalah memilih langkah yang menghasilkan keuntungan maksimal pada setiap langkah tanpa memperhitungkan langkah-langkah masa depan.

Berikut adalah beberapa langkah yang diambil oleh program saat membuat keputusan:

a. Mengambil diamond dengan Nilai Tinggi

   Bot akan memprioritaskan pengambilan diamond dengan nilai yang lebih tinggi. Ini dilakukan dengan cara memilih diamond yang terletak di sekitar posisi bot dan memiliki nilai lebih besar dibandingkan dengan diamond lainnya. Pemilihan ini tidak mempertimbangkan jarak jauh, tetapi berfokus pada peluang untuk mendapatkan diamond yang memberikan nilai lebih tinggi secara langsung.

b. Menentukan Lokasi Terdekat

   Pada setiap giliran, bot menghitung jarak ke berbagai diamond yang ada di peta dan memilih langkah yang membawa bot lebih dekat ke lokasi dengan banyak diamond dalam jangkauan. Strategi ini meminimalkan waktu yang dihabiskan untuk berpindah antar lokasi dan mengoptimalkan peluang untuk mengumpulkan lebih banyak diamond dalam waktu yang singkat.

c. Penggunaan Teleporter

   Jika ada teleporter yang dapat digunakan untuk berpindah dari satu lokasi ke lokasi lainnya dengan cepat, bot akan mempertimbangkan penggunaan teleporter tersebut untuk menghemat waktu. Penggunaan teleporter diutamakan jika dapat mengurangi waktu perjalanan yang diperlukan untuk mengumpulkan diamond lebih banyak, dengan tetap memastikan bot dapat kembali ke markas sebelum waktu habis.

d. Kembali ke Markas Jika Inventaris Penuh atau Waktu Habis

   Jika bot merasa inventarisnya hampir penuh atau waktu permainan hampir habis, bot akan memilih untuk segera kembali ke markas. Keputusan ini diambil berdasarkan analisis kondisi inventaris dan sisa waktu yang ada. Program ini juga memastikan bahwa perjalanan kembali ke markas dilakukan dengan cara yang paling efisien untuk menghindari waktu yang terbuang.

2.2.2 Menjalankan Bot Program

Untuk menjalankan bot, fungsi next_move() akan dipanggil setiap giliran permainan. Fungsi ini akan menganalisis kondisi sekitar bot dan menentukan arah gerakan yang optimal. Berdasarkan kondisi yang ada, bot akan memutuskan untuk melangkah ke posisi yang lebih dekat dengan diamond atau memilih jalur yang memungkinkan ia kembali ke markas dengan aman.

Bot juga memiliki mekanisme untuk memeriksa apakah ada ancaman atau rintangan di jalur yang akan diambil, dan jika ditemukan, ia akan mencoba mencari jalur alternatif yang lebih aman atau lebih efisien. Bot ini menggunakan prinsip greedy choice property, yaitu membuat keputusan yang mengarah pada pilihan terbaik pada setiap langkah, tanpa mempertimbangkan langkah-langkah selanjutnya.

2.2.3 Struktur Program

Program ini terdiri dari beberapa komponen yang bekerjasama untuk mengimplementasikan algoritma greedy. Fungsi utama program adalah sebagai berikut:

- Fungsi Pemindaian Peta

  Fungsi ini bertanggung jawab untuk memindai peta permainan, mengidentifikasi posisi diamond, dan mengelompokkan informasi terkait posisi tersebut. Ini memungkinkan bot untuk mengetahui lokasi yang bisa dijadikan target untuk pengumpulan diamond.
- Fungsi Perhitungan Jarak

  Fungsi ini digunakan untuk menghitung jarak antara posisi saat ini dengan posisi tujuan, baik itu diamond atau markas. Jarak ini dihitung menggunakan rumus Euclidean atau Manhattan, tergantung pada jenis peta permainan.
- Fungsi Keputusan Greedy

  Fungsi ini melakukan analisis terhadap peta dan memilih langkah terbaik berdasarkan kondisi lokal yang ada. Fungsi ini mengevaluasi semua kemungkinan langkah dan memilih yang memberikan keuntungan maksimal dalam hal jumlah diamond yang bisa dikumpulkan atau jarak yang bisa ditempuh dalam waktu yang tersedia.

- Fungsi next_move()

  Fungsi ini dipanggil setiap giliran permainan untuk menentukan arah gerakan bot. Fungsi ini menerima kondisi permainan saat ini, seperti posisi diamond, posisi bot, dan waktu yang tersisa, dan mengembalikan arah gerakan selanjutnya dalam bentuk tuple (dx, dy). Tuple ini menunjukkan pergerakan bot pada grid permainan, di mana dx dan dy adalah perubahan posisi dalam sumbu horizontal dan vertikal.

2.2.4 Penanganan Waktu dan Inventaris

Program ini mengatur waktu permainan dengan cermat. Bot mengelola waktu dengan memprioritaskan pengumpulan diamond pada awal permainan dan merencanakan rute pulang ke markas ketika waktu hampir habis. Jika bot mendekati batas waktu, ia akan memilih jalur yang lebih langsung menuju markas, menghindari area dengan diamond yang tidak dapat dijangkau sebelum waktu habis.

2.2.5 Penyempurnaan Algoritma

Untuk meningkatkan efisiensi algoritma, bot dapat ditingkatkan dengan teknik seperti backtracking atau dynamic programming untuk menangani kasus yang lebih kompleks di mana keputusan greedy mungkin tidak menghasilkan solusi optimal. Dengan penambahan teknik-teknik ini, bot dapat mengeksplorasi lebih banyak kemungkinan langkah yang lebih menguntungkan, namun tetap mempertahankan kecepatan eksekusi yang tinggi.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Proses mapping merupakan tahap awal dalam implementasi bot permainan yang berfungsi untuk memetakan area pencarian diamond di dalam peta permainan. Pada tahap ini, area pencarian dibatasi dengan menggunakan radius tertentu dari markas bot, yang bertujuan untuk memastikan bahwa bot tidak bergerak terlalu jauh dan tetap fokus pada area yang menguntungkan. Fungsi get_all_diamonds_within_limit(limit) digunakan untuk mengambil informasi mengenai semua diamond yang berada dalam radius tersebut. Parameter limit ini ditentukan berdasarkan jarak yang dapat dicapai oleh bot dalam waktu yang terbatas, serta kapasitas inventaris yang ada.

Pada implementasi awal, radius pencarian diamond diatur dengan nilai default 9. Namun, jika tidak ada diamond yang ditemukan dalam radius tersebut, program secara otomatis akan memperluas radius pencarian untuk mencakup area yang lebih luas. Hal ini memungkinkan bot untuk mengeksplorasi lebih banyak wilayah, terutama ketika area sekitar markas tidak kaya akan sumber daya. Dengan memperluas pencarian, bot memiliki kesempatan lebih besar untuk menemukan diamond yang lebih banyak atau lebih bernilai, tanpa membuang-buang waktu dan sumber daya untuk mencari di area yang tidak produktif.

Pendekatan ini sangat berguna untuk menjaga keseimbangan antara efisiensi dan efektivitas pencarian diamond. Proses mapping juga memastikan bahwa bot tetap dalam kendali dan tidak menyimpang terlalu jauh dari markas, yang memungkinkan bot untuk kembali tepat waktu dengan jumlah diamond yang optimal.

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam pengembangan bot ini, beberapa alternatif strategi greedy telah dipertimbangkan untuk meningkatkan efisiensi pengumpulan diamond. Setiap strategi memiliki keunggulan dan kekurangan yang harus dievaluasi dengan cermat berdasarkan karakteristik peta dan distribusi diamond yang ada. Berikut adalah beberapa alternatif yang dipertimbangkan:

- **Greedy berdasarkan nilai diamond tertinggi**

  Pada pendekatan ini, bot selalu memilih diamond yang memiliki nilai tertinggi yang terdekat dengan posisinya. Strategi ini sangat efektif jika diamond bernilai tinggi selalu berada dekat satu sama lain, namun kurang optimal ketika diamond tersebar secara acak di seluruh peta. Pemilihan berdasarkan nilai tertinggi menjadikan bot lebih fokus pada pengumpulan diamond dengan nilai maksimal, tetapi mungkin mengabaikan kesempatan untuk mengumpulkan lebih banyak diamond jika nilai-nilai rendah berada lebih dekat.
- **Greedy berdasarkan jarak terdekat ke diamond tanpa mempertimbangkan teleporter**

  Dalam strategi ini, bot memilih langkah berdasarkan jarak Manhattan terdekat menuju diamond tanpa mempertimbangkan penggunaan teleporter. Hal ini sangat berguna untuk peta yang relatif kecil dan tidak memiliki fitur teleporter, namun pada peta yang lebih besar dan lebih kompleks, strategi ini dapat mengakibatkan bot melakukan perjalanan jauh yang tidak efisien, menghabiskan lebih banyak waktu daripada yang seharusnya diperlukan.
- **Greedy berdasarkan rerata jarak ke semua diamond dalam radius tertentu**

  Pendekatan ini memilih langkah berdasarkan posisi yang memiliki rata-rata jarak terdekat ke banyak diamond yang ada dalam radius pencarian tertentu. Strategi ini memberikan hasil yang lebih stabil karena mempertimbangkan keseluruhan distribusi diamond, menghindari bot untuk hanya fokus pada diamond bernilai tinggi di lokasi yang sangat jauh. Meskipun strategi ini lebih stabil, hasilnya mungkin kurang optimal dalam hal perolehan diamond yang sangat bernilai tinggi karena bot tidak secara langsung menuju ke diamond-diamond tersebut.
- **Greedy adaptif dengan penggunaan teleporter**

  Strategi ini mengadaptasi pendekatan greedy dengan mempertimbangkan penggunaan teleporter jika jarak yang ditempuh menggunakan teleporter lebih singkat daripada jalur langsung. Penggunaan teleporter dapat menghemat banyak waktu, terutama pada peta besar dan kompleks dengan diamond yang tersebar jauh. Dengan memanfaatkan teleporter secara strategis, bot dapat menghemat perjalanan yang memakan waktu dan mempercepat pengumpulan diamond.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setiap alternatif strategi greedy memiliki tingkat efisiensi dan efektivitas yang berbeda-beda, tergantung pada karakteristik peta dan distribusi diamond yang ada. Berikut adalah analisis lebih lanjut tentang efektivitas dan efisiensi masing-masing strategi:

- **Strategi berdasarkan nilai tertinggi**

  Strategi ini sangat efisien apabila diamond bernilai tinggi berada dekat satu sama lain. Namun, apabila distribusi diamond bersifat acak, strategi ini cenderung kurang optimal. Bot akan terfokus pada diamond bernilai tinggi yang terdekat tanpa mempertimbangkan total jumlah diamond yang dapat dikumpulkan. Hal ini dapat menyebabkan waktu yang terbuang untuk berpindah ke tempat yang jauh hanya untuk mengumpulkan diamond yang nilainya tidak optimal.

- **Strategi tanpa mempertimbangkan teleporter**

  Meskipun strategi ini sederhana, ia cenderung tidak efisien dalam peta yang besar karena tidak mempertimbangkan penggunaan jalur tercepat, seperti teleporter. Pada peta yang lebih besar dan lebih kompleks, bot yang hanya mengandalkan perhitungan jarak terdekat tanpa memperhitungkan elemen-elemen seperti teleporter bisa menghabiskan waktu yang lebih lama daripada yang seharusnya.

- **Strategi rerata jarak**

  Strategi ini memberikan hasil yang lebih stabil dan seimbang karena mempertimbangkan distribusi diamond di seluruh peta. Meskipun ini tidak selalu memberikan solusi yang paling efisien dalam hal waktu atau jumlah diamond yang dikumpulkan, namun strategi ini memastikan bahwa bot tidak hanya mengejar diamond bernilai tinggi yang terletak sangat jauh, melainkan juga memastikan bahwa perjalanan tetap terorganisir dan bot tetap berada dalam radius yang dapat dijangkau.

- **Strategi dengan penggunaan teleporter**

  Penggunaan teleporter secara strategis memungkinkan bot untuk menghemat waktu secara signifikan, terutama dalam peta yang lebih besar dengan banyak diamond yang tersebar di berbagai lokasi jauh. Strategi ini cenderung memberikan hasil terbaik dalam peta yang rumit dan besar karena mengoptimalkan waktu yang dibutuhkan untuk berpindah dari satu titik ke titik lainnya.

3.4 Strategi Greedy yang Dipilih

Setelah menganalisis berbagai alternatif strategi greedy, keputusan dibuat untuk memilih kombinasi dari strategi rerata jarak ke diamond dalam area terbatas dengan mempertimbangkan penggunaan teleporter. Pendekatan ini terbukti paling adaptif terhadap variasi peta dan distribusi diamond yang ada.

Dengan menggunakan strategi ini, bot tidak hanya mengumpulkan diamond yang ada di sekitar secara efisien, tetapi juga tetap dapat memanfaatkan elemen-elemen tambahan seperti teleporter untuk mempercepat perjalanan ke tempat yang lebih jauh. Strategi ini juga memastikan bahwa bot dapat kembali ke markas tepat waktu dengan jumlah diamond yang maksimal, mengingat faktor waktu dan inventaris yang harus dikelola dengan hati-hati.

Kombinasi strategi ini menunjukkan keseimbangan antara efisiensi langkah, efektivitas perolehan diamond, dan keamanan perjalanan bot. Dengan pendekatan ini, bot dapat beradaptasi dengan perubahan dinamika peta dan mencari jalan keluar yang terbaik, baik itu untuk memaksimalkan jumlah diamond yang dikumpulkan atau untuk mengoptimalkan waktu tempuh menuju markas.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

```
from typing import Optional, List, Tuple

from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
from time import sleep

class BotNation(BaseLogic):
    """
    Bot strategis yang fokus pada area sekitar base.
    Mengumpulkan diamond terdekat dan kembali ketika inventory mencapai batas.
    """

    def __init__(self):
        # Radius maksimum pencarian dari base
        self.search_radius = 7

    def next_move(self, board_bot: GameObject, board: Board):
        """Method utama untuk pengambilan keputusan pergerakan bot"""
        self.game_board = board
        self.player = board_bot

        # Cek waktu darurat - kurang dari 2 detik tersisa
        if (self.player.properties.milliseconds_left // 1000 < 2):
            print("OH NOOOOOOO")

        # Kembali ke base jika waktu hampir habis
        if (self.player.properties.milliseconds_left // 1000 <= self.calc_teleport_dist(self.player.position,
self.player.properties.base) + 2):
            print("byee zzz")
            return self.head_to_base()

        # Ambil diamond dalam area pencarian
        nearby_gems = self.get_gems_in_radius(self.search_radius)
        if len(nearby_gems) == 0:
            nearby_gems = self.get_gems_in_radius(self.search_radius + 2)
```

```
        if len(nearby_gems) == 0:
            return self.move_to_center()

        # Kasus khusus hanya ada 2 diamond
        if len(nearby_gems) == 2:
            closest_gem = min(nearby_gems, key=lambda gem: self.calc_teleport_dist(self.player.position,
gem.position))
            return self.nav_with_teleport(closest_gem.position)

        # Cari diamond dalam jangkauan
        all_gems = self.game_board.diamonds
        gems_one_step = self.find_gems_within_steps(1, all_gems)
        gems_two_steps = self.find_gems_within_steps(2, all_gems)

        # Filter berdasarkan kapasitas inventory
        gems_one_step = list(filter(lambda gem: self.player.properties.diamonds + gem.properties.points <
self.player.properties.inventory_size, gems_one_step))
        gems_two_steps = list(filter(lambda gem: self.player.properties.diamonds + gem.properties.points <
self.player.properties.inventory_size, gems_two_steps))

        # Prioritaskan diamond dalam satu langkah
        if len(gems_one_step) > 0:
            # Pilih diamond bernilai tinggi (2 poin) terlebih dahulu
            for gem in gems_one_step:
                if (gem.properties.points == 2):
                    return self.nav_with_teleport(gem.position)

            optimal_gem = min(gems_one_step, key=lambda gem: self.calc_avg_gem_dist(gem.position,
nearby_gems))
            return self.nav_with_teleport(optimal_gem.position)

        # Cek diamond dalam dua langkah
        elif len(gems_two_steps) > 0:
            # Pilih diamond bernilai tinggi (2 poin) terlebih dahulu
            for gem in gems_two_steps:
                if (gem.properties.points == 2):
                    return self.nav_with_teleport(gem.position)

            optimal_gem = min(gems_two_steps, key=lambda gem: self.calc_avg_gem_dist(gem.position,
nearby_gems))
            return self.nav_with_teleport(optimal_gem.position)

        # Logika keputusan untuk kembali ke base
        if (self.player.properties.diamonds >= self.player.properties.inventory_size - 1):
```

```
            return self.head_to_base()
        elif (self.calc_teleport_dist(self.player.position, self.player.properties.base) <= 2 and
self.player.properties.diamonds >= 3):
            return self.head_to_base()
        elif (self.calc_teleport_dist(self.player.position, self.player.properties.base) <= 1 and
self.player.properties.diamonds >= 1):
            return self.head_to_base()

        # Cari posisi optimal dengan jarak rata-rata minimum ke diamond
        optimal_pos = self.find_best_adjacent_pos(self.player.position, nearby_gems)
        print(f"OTEWE KE {optimal_pos}")
        return self.nav_with_teleport(optimal_pos)

    def get_teleport_pair(self) -> Tuple[GameObject, GameObject]:
        """Mengambil kedua objek teleporter dari papan permainan"""
        teleports = [obj for obj in self.game_board.game_objects if obj.type == "TeleportGameObject"]
        if len(teleports) != 2:
            return None, None
        return teleports[0], teleports[1]

    def get_nearest_teleport(self):
        """Mencari teleporter yang paling dekat dengan posisi saat ini"""
        curr_pos = self.player.position
        tp1, tp2 = self.get_teleport_pair()
        nearest_tp = tp1 if (self.calc_manhattan_dist(curr_pos, tp1.position) <
self.calc_manhattan_dist(curr_pos, tp2.position)) else tp2
        return nearest_tp

    def calc_manhattan_dist(self, pos_a: Position, pos_b: Position) -> int:
        """Menghitung jarak Manhattan antara dua posisi"""
        dx = abs(pos_a.x - pos_b.x)
        dy = abs(pos_a.y - pos_b.y)
        return dx + dy

    def calc_teleport_dist(self, pos_a: Position, pos_b: Position) -> int:
        """Menghitung jarak terpendek dengan mempertimbangkan penggunaan teleporter"""
        tp1, tp2 = self.get_teleport_pair()

        nearest_tp = tp1 if (self.calc_manhattan_dist(pos_a, tp1.position) < self.calc_manhattan_dist(pos_a,
tp2.position)) else tp2
        other_tp = tp1 if (self.calc_manhattan_dist(pos_a, tp1.position) > self.calc_manhattan_dist(pos_a,
tp2.position)) else tp2

        dist_to_tp = self.calc_manhattan_dist(pos_a, nearest_tp.position)
        tp_to_dest = self.calc_manhattan_dist(other_tp.position, pos_b)
```

```
    direct_dist = self.calc_manhattan_dist(pos_a, pos_b)

    return min(direct_dist, dist_to_tp + tp_to_dest)

def nav_to_target(self, target: Position) -> Tuple[int, int]:
    """Navigasi menuju target tanpa mempertimbangkan teleporter"""
    dx = target.x - self.player.position.x
    dy = target.y - self.player.position.y

    move_dir = (0, 0)

    if abs(dx) > abs(dy):
        move_dir = (1 if dx > 0 else -1, 0)
    else:
        move_dir = (0, 1 if dy > 0 else -1)

    curr_pos = self.player.position

    # Cek batas area
    within_bounds = (curr_pos.x + move_dir[0] >= 0 and
                     curr_pos.y + move_dir[1] >= 0 and
                     curr_pos.x + move_dir[0] < self.game_board.width + 1 and
                     curr_pos.y + move_dir[1] < self.game_board.height + 1)

    if within_bounds:
        return move_dir
    else:
        return (move_dir[0] * (-1), move_dir[1] * (-1))

def nav_with_teleport(self, target: Position) -> Tuple[int, int]:
    """Navigasi menuju target dengan mempertimbangkan optimasi teleporter"""
    curr_pos = self.player.position

    if (self.calc_teleport_dist(curr_pos, target) == self.calc_manhattan_dist(curr_pos, target)):
        return self.nav_to_target(target)
    elif (self.calc_teleport_dist(curr_pos, target) < self.calc_manhattan_dist(curr_pos, target)):
        nearest_tp = self.get_nearest_teleport()
        return self.nav_to_target(nearest_tp.position)

def head_to_base(self) -> Tuple[int, int]:
    """Navigasi kembali ke base rumah"""
    return self.nav_with_teleport(self.player.properties.base)

def move_to_center(self) -> Tuple[int, int]:
```

```
        """Bergerak menuju pusat papan"""
        center_pos = Position(self.game_board.height // 2, self.game_board.width // 2)
        return self.nav_with_teleport(center_pos)

    def get_opponents(self) -> list[GameObject]:
        """Mendapatkan daftar semua bot musuh"""
        opponents = self.game_board.bots
        opponents.remove(self.player)
        return opponents

    def find_nearest_opponent(self) -> GameObject:
        """Mencari bot musuh yang paling dekat"""
        nearest = None
        min_dist = 30

        enemies = self.get_opponents()
        for enemy in enemies:
            enemy_dist = self.calc_teleport_dist(self.player.position, enemy.position)
            if enemy_dist < min_dist:
                min_dist = enemy_dist
                nearest = enemy
        return nearest

    def get_bot_idx(self, target_bot: GameObject) -> int:
        """Mendapatkan indeks bot tertentu dalam daftar bot"""
        bots = self.game_board.bots
        for i in range(len(bots)):
            if bots[i] == target_bot:
                return i

    def get_gems_in_radius(self, radius):
        """Mendapatkan semua diamond dalam radius tertentu dari base"""
        gems = self.game_board.diamonds
        return list(filter(lambda gem: self.calc_teleport_dist(gem.position, self.player.properties.base) <=
radius, gems))

    def calc_avg_gem_dist(self, pos: Position, gems: list[GameObject]):
        """Menghitung jarak rata-rata berbobot dari posisi ke semua diamond"""
        total_dist = 0

        for gem in gems:
            # Bobot berdasarkan poin kuadrat dan ambil akar untuk optimasi
            weighted_dist = (self.calc_teleport_dist(gem.position, pos) * gem.properties.points *
gem.properties.points) ** (0.25)
            total_dist += weighted_dist
```

```
        return total_dist / len(gems)

    def find_best_adjacent_pos(self, pos: Position, gems: list[GameObject]):
        """Mencari posisi bersebelahan dengan jarak rata-rata minimum ke diamond"""
        adjacent_positions = []
        adjacent_positions.append(Position(pos.y + 1, pos.x))
        adjacent_positions.append(Position(pos.y - 1, pos.x))
        adjacent_positions.append(Position(pos.y, pos.x + 1))
        adjacent_positions.append(Position(pos.y, pos.x - 1))

        return min(adjacent_positions, key=lambda p: self.calc_avg_gem_dist(p, gems))

    def find_gems_within_steps(self, steps: int, gems: list[GameObject]):
        """Mencari diamond dalam n langkah dari posisi saat ini"""
        return list(filter(lambda gem: self.calc_teleport_dist(self.player.position, gem.position) <= steps,
gems))
```

4.1.2 Penjelasan Alur Program

Kode program diatas merupakan implementasi Algoritma Greedy dalam logika permainan Etimo Diamond, pergerakan bot dalam permainan ini memiliki tujuan untuk mengumpulkan diamond di dalam sebuah map yang berbasis grid. Bot ini diberi nama **BotNation** dan mewarisi **BaseLogic**, dengan strategi utama yaitu mengumpulkan diamond di sekitar base lalu kembali sebelum waktu habis atau inventory penuh. Ketika fungsi **next_move()** dipanggil, bot akan memeriksa apakah waktu yang tersisa cukup untuk kembali ke base. Jika waktu yang tersisa sedikit atau tidak cukup, maka bot akan langsung kembali ke base dengan fungsi **head_to_base()** dengan perhitungan jarak tercepat, dan mempertimbangkan kemungkinan penggunaan teleporter.

Bot juga mencari diamond dalam radius 7 grid di sekitar base, dengan memprioritaskan diamond bernilai tinggi (2 poin), dan menghindari mengambil diamond jika kapasitas inventory hampir penuh (>=4). Bot memanfaatkan fungsi **calc_teleport_dist()** untuk menghitung jarak efektif dengan memperhitungkan penggunaan teleporter, dan memilih jalur tercepat antara berjalan langsung atau melalui teleporter. Jika tidak ada diamond terdekat, bot akan bergerak ke titik tengah *board* untuk kemungkinan posisi strategis. Selain itu, bot juga mampu mencari posisi optimal di sekitar dirinya yang memberikan rata-rata jarak terbaik ke diamond menggunakan fungsi **calc_avg_gem_dist()** dan **find_best_adjacent_pos().**

Secara keseluruhan, bot ini dirancang untuk bergerak efisien dan cerdas dalam mengambil keputusan berdasarkan kondisi waktu, inventory, dan posisi diamond, serta fleksibel dalam memanfaatkan fitur-fitur permainan seperti teleporter.

4.2 Struktur Data yang Digunakan

Dalam program ini terdapat beberapa struktur data utama yang digunakan dalam pembuatan algoritma secara langsung. Struktur data tersebut adalah sebagai berikut :

1. Objek GameObject

   Objek GameObject merupakan entitas utama yang ada di papan permainan (Board). Bot, diamond, teleporter, dan red button semuanya direpresentasikan sebagai turunan dari GameObject. Atribut penting dalam GameObject yang digunakan dalam kode ini meliputi:

   - Position : Menentukan lokasi objek di papan permainan.
   - Type : Menentukan jenis objek, seperti TeleportGameObject, DiamondGameObject, atau BotGameObject.
   - Properties :
     - Diamond : Poin (nilai diamond) dan position.
     - Bot : Atribut seperti diamonds, inventory_size, milliseconds_left, dan base.
     - Teleporter dan red button : Position yang digunakan.

2. Objek Board

   Objek Board merupakan representasi dari papan permainan yang digunakan oleh bot untuk memahami lingkungan sekitar. Atribut penting dalam Board yang dimanfaatkan dalam kode ini meliputi:

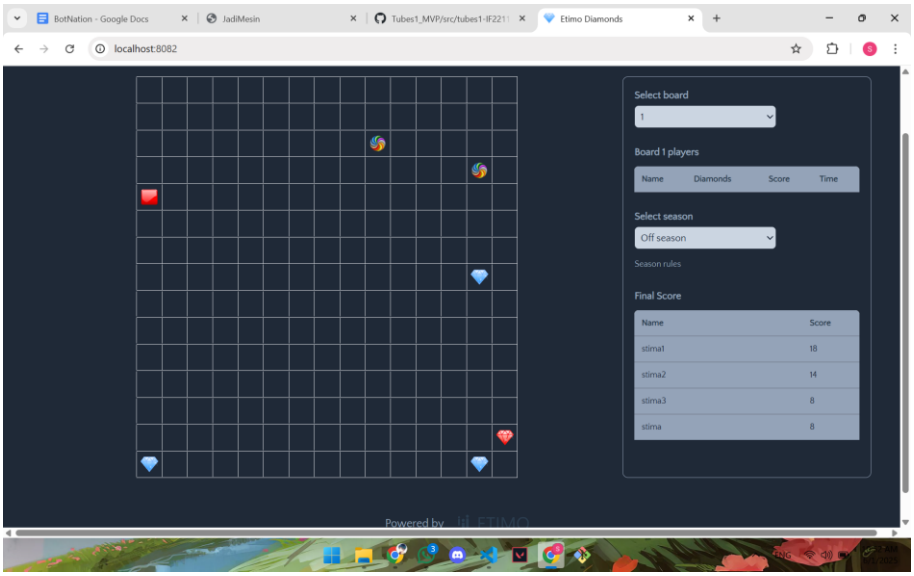
   - height dan width : Ukuran dimensi papan permainan.
   - game_objects: List dari seluruh GameObject yang ada di dalam papan permainan, termasuk diamond, teleport, dan red button.
   - bot : List dari semua bot yang bermain pada satu sesi permainan, termasuk bot kita sendiri.
   - diamonds: List dari semua diamond yang ada di papan permainan

4.3 Pengujian Program

4.3.1 Skenario Pengujian

- Percobaan menjalankan 4 bot sekaligus dalam satu waktu
- Percobaan Tackle dan Defend
- Percobaan Teleporter

4.3.2 Hasil Pengujian dan Analisis



Gambar 7. Hasil Pengujian

Dari hasil percobaan Skenario pertama, kami mendapatkan hasil seperti gambar diatas dengan skor maksimum sebesar 18 Poin, minimum sebesar 8 poin, dengan rata - rata 12 poin. Dalam percobaan ini bot dijalankan selama 60s sesuai dengan waktu default program.

Pada skenario kedua , hasil yang didapatkan adalah ketika bot membawa diamond dia akan cenderung bergerak untuk menjauhi musuh di sekitarnya, bot mendapatkan info bot di sekitarnya dengan memanfaatkan fungsi **find_nearest_opponents()**, dan ketika bot sedang tidak memiliki diamond di dalam inventory maka bot akan melakukan tackle kepada bot lain yang membawa diamond.

Pada skenario ketiga, hasil yang didapatkan adalah ketika tidak ada diamond di sekitar diamond, namun ada teleporter bot dapat memanfaatkan teleporter untuk mencari diamond yang berada disekitar teleporter.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Ada banyak kejadian yang mungkin terjadi pada permainan diamonds ini. Untuk mendapatkan kemenangan pada permainan ini Algoritma Greedy yang diimplementasikan harus memuat langkah yang terbaik dalam mengumpulkan poin selama permainan. Dengan melakukan pertimbangan terhadap efisiensi, efektivitas, dan risiko dari kumpulan alternatif algoritma greedy yang ada, kami memutuskan untuk memilih strategi rerata jarak ke diamond dalam area terbatas dengan mempertimbangkan penggunaan teleporter sebagai solusi terbaik. Hal tersebut juga didukung oleh beberapa percobaan yang kami lakukan. Algoritma greedy memilih diamond berdasarkan rerata jarak lebih sering memenangkan pertandingan dibandingkan alternatif solusi lainnya.

5.2 Saran

Proses eksplorasi dan pemahaman yang mendalam terhadap permasalahan adalah hal yang perlu diprioritaskan sebelum pengambilan keputusan dan implementasi. Ada banyak *test case* atau kombinasi unika yang mungkin terjadi pada suatu persoalan khususnya pada permainan Diamonds. Mungkin saja kombinasi-kombinasi unik dan aneh tersebut akan menjadi parameter yang krusial untuk memutuskan algoritma greedy yang akan digunakan.

LAMPIRAN

A. Repository Github

Link Repository tugas besar Strategi Algoritma kelompok 4 “BotNation” :

https://bit.ly/Github_Tubes1_BotNation

B. Video Youtube

Link Video Youtube tugas besar Strategi Algoritma kelompok 4 “BotNation” :

https://bit.ly/Tubes_1BotNation

DAFTAR PUSTAKA

[1] T. Cormen H., C. Leiserson E., R. Rivest R., and C. Stein, *Introduction to Algoritms*, 3rd ed. MIT, 2009.