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 5 (BOTYY)

Jesika Filosovi Br P-A	123140044
Nabila Ramadhani Mujahidin	123140062
Willy Syifa Luthfia	123140071

Dosen Pengampu: Imam Eko wicaksono, S.Si., M.Si.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Diamonds adalah sebuah tantangan pemrograman di mana bot yang Anda buat akan bersaing dengan bot milik pemain lain. Setiap peserta akan memiliki satu bot dengan tujuan utama yaitu mengumpulkan sebanyak mungkin diamond. Tentu saja, proses pengumpulan diamond tidak akan mudah—akan ada berbagai rintangan yang membuat permainan ini lebih menantang dan menarik.

Untuk meraih kemenangan, setiap peserta perlu merancang dan menerapkan strategi yang efektif pada bot miliknya. Pada tugas pertama dalam mata kuliah Strategi Algoritma ini, mahasiswa diminta untuk mengembangkan sebuah bot yang nantinya akan dipertandingkan satu sama lain. Strategi yang harus digunakan dalam pengembangan bot ini adalah **strategi greedy**.

Program permainan Diamonds terdiri atas:

1. *Game engine,* yang secara umum berisi:
    a. Kode *backend* permainan, yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *frontend* dan program bot
    b. Kode *frontend* permainan, yang berfungsi untuk memvisualisasikan permainan
2. *Bot starter pack,* yang secara umum berisi:
    a. Program untuk memanggil API yang tersedia pada *backend*
    b. Program *bot logic* (bagian ini yang akan kalian implementasikan dengan algoritma *greedy* untuk bot kelompok kalian)
    c. Program utama (*main*) dan utilitas lainnya

Komponen-komponen dari permainan Diamonds antara lain:

1. **Diamonds**

Untuk 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.

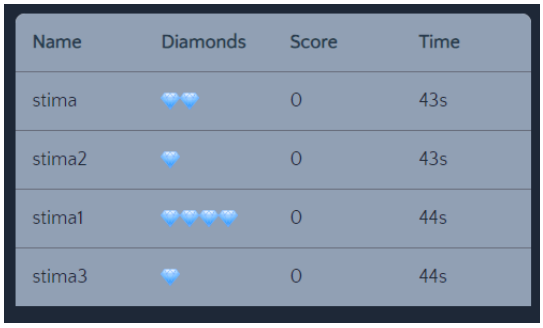
3. **Teleporters**

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.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy adalah metode penyelesaian masalah yang menggunakan pendekatan membuat pilihan terbaik pada setiap langkah lokal dengan harapan keputusan tersebut akan mengarah pada solusi optimal secara global . Pada setiap iterasi, algoritma ini memilih opsi yang memberikan manfaat maksimal saat itu juga, tanpa mempertimbangkan konsekuensi di masa depan atau meninjau kembali keputusan yang sudah diambil.

Agar pendekatan greedy memberikan solusi optimal, masalah yang diselesaikan harus memiliki dua properti utama:

1. **Greedy Choice Property**, yaitu solusi optimal dapat dicapai dengan membuat keputusan lokal yang optimal.
2. **Optimal Substructure**, yaitu solusi dari masalah dapat dibentuk dari solusi optimal submasalah-submasalah nya.

Langkah umum algoritma greedy dapat dirangkum sebagai berikut:

1. Inisialisasi solusi awal (biasanya kosong).
2. Selama solusi belum lengkap:
   - Pilih elemen terbaik menurut kriteria tertentu.
   - Tambahkan elemen tersebut ke solusi.
3. Kembalikan solusi akhir.

Contoh Penerapan Algoritma Greedy

Beberapa masalah klasik yang dapat diselesaikan secara optimal dengan algoritma greedy antara lain:

- **Minimum Spanning Tree (MST)** seperti pada algoritma Kruskal dan Prim [1].
- **Huffman Coding** untuk kompresi data.

- **Activity Selection Problem**, untuk memilih sebanyak mungkin aktivitas yang tidak saling tumpang tindih dalam rentang waktu tertentu.

Kelebihan dan Kekurangan

Kelebihan utama dari algoritma greedy adalah efisiensinya dalam hal waktu dan kesederhanaan implementasi. Namun, kekurangannya adalah tidak semua masalah dapat diselesaikan secara optimal menggunakan pendekatan ini. Jika properti greedy tidak terpenuhi, maka solusi yang dihasilkan bisa jauh dari optimal.

2.2 Cara Kerja Program

Program bot adalah perangkat lunak yang dirancang untuk melakukan tugas-tugas tertentu secara otomatis berdasarkan aturan atau algoritma yang sudah ditentukan. Secara umum, bot menerima input, memproses input tersebut menggunakan logika atau algoritma tertentu, kemudian menghasilkan output atau melakukan aksi yang sesuai.

Dalam konteks bot berbasis algoritma greedy, bot akan berupaya menyelesaikan masalah dengan memilih opsi terbaik secara lokal pada setiap langkah, dengan harapan bahwa pilihan lokal tersebut akan menghasilkan solusi global yang optimal atau mendekati optimal.

Secara garis besar, proses pembuatan algoritma hanya akan dilakukan pada folder **/game/logic.** Semua file diluar folder logic adalah file yang berfungsi sebagai komponen-komponen untuk menjalankan bot dan untuk berkomunikasi dengan website game

1. Cara Implementasi Program

Program bot ini dirancang untuk bermain dalam permainan *Diamonds* secara otomatis menggunakan pendekatan algoritma greedy. Tujuan utama dari bot adalah untuk mengumpulkan diamond sebanyak mungkin dalam waktu yang terbatas dengan memprioritaskan langkah-langkah yang memberikan nilai keuntungan tertinggi per satuan langkah (value per distance).

Alur Umum Kerja Bot

1. **Inisialisasi Informasi Permainan**

   Bot menerima input awal berupa peta permainan, posisi dirinya, lokasi semua objek penting (red diamond, blue diamond, red button, teleporter, base, dan musuh), serta status inventory.

2. **Pencarian Kandidat Solusi**

   Bot mengidentifikasi semua objek pada peta yang memungkinkan untuk diambil atau digunakan, kemudian menghitung jarak dari posisi bot ke masing-masing objek menggunakan Manhattan Distance.

3. **Perhitungan Value per Distance (Density)**

   Untuk setiap kandidat objek, bot menghitung skor rasio antara nilai poin dan jaraknya dari posisi bot. Rumus yang digunakan:

   $$\text{Density} = \frac{Poin\ Objek}{Jarak\ dari\ Bot\ ke\ Objek}$$

4. **Seleksi Obyek Terbaik**

   Bot memilih objek dengan nilai density tertinggi yang juga layak diambil (feasible), misalnya inventory belum penuh atau objek tidak berada dalam jalur tertutup.

5. **Pergerakan Menuju Obyek**

   Setelah objek tujuan dipilih, bot merencanakan langkah demi langkah untuk menuju ke lokasi tersebut dan mengambilnya.

6. **Pengulangan Proses**

   Langkah 2 sampai 5 terus dilakukan selama permainan masih berlangsung dan waktu masih tersisa. Di akhir permainan, bot juga mengevaluasi apakah masih cukup waktu untuk kembali ke base dan menyimpan diamond.

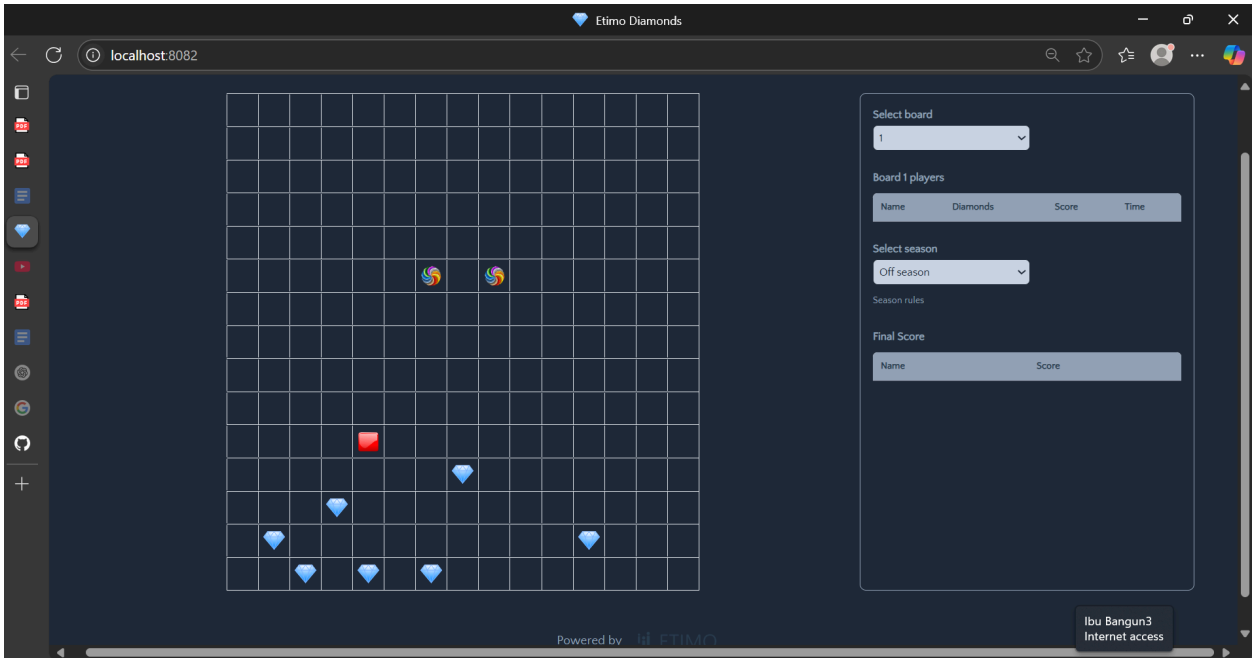
Strategi Pendukung

- Bot menghindari tujuan yang tidak bisa diakses tepat waktu atau tidak memberikan keuntungan optimal.
- Jika red diamond menjadi target, bot memastikan inventory tidak penuh (maksimum 4).
- Bot akan mengabaikan objek dengan nilai density yang rendah jika ada opsi yang lebih efisien.
- Saat mendekati akhir permainan, bot mengubah prioritas untuk **kembali ke base tepat waktu**.

Dengan mengikuti strategi ini, program bot diharapkan dapat membuat keputusan cerdas secara lokal yang mengarah pada hasil yang mendekati optimal secara keseluruhan, sesuai dengan prinsip algoritma greedy.

2. Menjalankan Bot Program

Sebelum memainkan game ini, pemain perlu memasang ketiga aplikasi ini yaitu Node.js, Docker desktop, dan Yarn. Pada Docker, pemain juga harus membuat database yang diperlukan oleh aplikasi berjalan pada Docker tersebut. Setelah semuanya selesai, program perlu melakukan build terlebih dahulu sebelum dijalankan dengan masuk ke website “http://localhost:8082/ untuk memulai permainan tersebut. Beginilah bentuk dari program yang akan dimainkan.



Untuk menjalankan bot pada halaman web ini, terdapat 2 cara yang bisa dilakukan. Namun sebelum itu, pastikan pemain sudah memasang dependencies yang dibutuhkan oleh program. Cara pertama untuk menggunakan bot adalah dengan menjalankan command ini (nama, email, password bisa diganti).

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

Terdapat cara lain yang bisa digunakan untuk memainkan Diamonds dengan beberapa bot sekaligus dan dengan strategi yang berbeda-beda.

Untuk Windows: ./run-bots.bat

Untuk Linux: ./run-bots.sh

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Berdasarkan studi pustaka yang telah dilakukan pada landasan teori, diketahui bahwa algoritma greedy bekerja dengan mencari suatu himpunan bagian (S) dari himpunan kandidat (C). Dalam hal ini, himpunan solusi S harus memenuhi dua kriteria utama:

1. S merupakan solusi yang valid, dan
2. S harus dioptimalkan melalui suatu fungsi objektif.

Oleh karena itu, untuk dapat menerapkan algoritma greedy dalam permasalahan game Diamonds, terlebih dahulu perlu dilakukan identifikasi terhadap elemen-elemen dalam permainan yang dapat dipetakan ke dalam komponen-komponen algoritma greedy. Berikut adalah pemetaan yang dimaksud:

1. Himpunan Kandidat C

Himpunan kandidat mencakup seluruh koordinat objek dalam game yang dapat menjadi tujuan bot. Objek-objek tersebut meliputi:

- Blue diamond
- Red diamond
- Red button
- Teleporter
- Base bot
- Bot musuh
- Base musuh

Selain itu, elemen penting lainnya yang perlu diperhatikan adalah jarak antara bot dengan setiap koordinat tujuan, karena hal ini memengaruhi efektivitas langkah yang diambil.

2. Himpunan Solusi (S)

Himpunan solusi berisi satu atau lebih koordinat yang memberikan keuntungan terbesar ketika dikunjungi oleh bot. Dalam pendekatan greedy, S dapat terdiri dari:

- Satu koordinat saja (solusi tunggal), atau
- Beberapa koordinat yang dilalui secara berurutan (multi-langkah), tergantung pada strategi greedy yang digunakan.

3. Fungsi Solusi

Fungsi ini berfungsi untuk memeriksa apakah solusi telah ditemukan. Umumnya, fungsi ini digunakan menjelang akhir permainan, untuk mengevaluasi apakah waktu yang tersisa masih cukup bagi bot untuk:

- Menjangkau koordinat tujuan, dan
- Kembali ke base sebelum waktu habis.

Fungsi solusi jarang digunakan pada awal permainan karena proses seleksi kandidat di fungsi seleksi telah menyaring langkah-langkah yang tidak mungkin menjadi solusi.

4. Fungsi Seleksi (Selection Function)

Fungsi seleksi bertanggung jawab untuk memilih kandidat terbaik berdasarkan strategi greedy tertentu. Setiap pendekatan greedy yang berbeda akan menggunakan fungsi seleksi yang berbeda pula. Penjelasan lebih rinci mengenai variasi fungsi seleksi akan dibahas pada bagian alternatif solusi.

5. Fungsi Kelayakan (Feasibility Function)

Fungsi ini digunakan untuk memastikan apakah kandidat yang dipilih layak dimasukkan ke dalam solusi. Dalam konteks game Diamonds, fungsi ini berperan menentukan apakah sebuah objek dapat menjadi tujuan bot.

Contoh sederhana:

Jika bot sudah menyimpan 4 diamond, maka ia tidak dapat mengambil red diamond (karena kapasitas penuh). Fungsi kelayakan akan mencegah bot menuju lokasi tersebut, sehingga menghindari kemungkinan terjadinya bug seperti bot berputar-putar di sekitar objek atau melakukan invalid move.

6. Fungsi Objektif (Objective Function)

Fungsi objektif berperan dalam mengoptimalkan pemilihan kandidat dari himpunan C menjadi solusi S. Dalam permainan ini, tujuan fungsi objektif adalah memaksimalkan keuntungan berdasarkan nilai objek dan jarak tempuh. Semakin dekat jaraknya dan semakin besar poin dari objek tersebut, maka semakin tinggi keuntungannya. Karena bot hanya dapat bergerak secara vertikal dan horizontal (tidak diagonal), maka jarak dihitung menggunakan rumus Manhattan Distance:

$$\text{Distance} = |x1 - x2| + |y1 - y2|$$

Dengan:

- (x1,y1): posisi bot saat ini
- (x2,y2): posisi objek tujuan

Selanjutnya, keuntungan dari suatu objek dapat dihitung secara sederhana dengan:

$$\text{Keuntungan} = \frac{Poin\ Objek}{Jarak}$$

Namun demikian, fungsi objektif ini dapat disesuaikan untuk masing-masing alternatif strategi greedy yang digunakan. Penjelasan mengenai fungsi objektif untuk masing-masing pendekatan akan dibahas lebih lanjut pada bagian berikutnya.

3.2 Eksplorasi Alternatif Solusi Greedy

1. **Greedy Berdasarkan Nilai Maksimal (Greedy by Highest Point)**

   Bot akan selalu memilih diamond dengan nilai tertinggi terlebih dahulu, tanpa mempertimbangkan jarak tempuh.

   Prioritas:

   - Red diamond (2 poin) selalu diprioritaskan dibanding blue diamond (1 poin), tanpa memedulikan jarak.

   Kelebihan:

   - Mudah diimplementasikan.
   - Potensi nilai per langkah tinggi jika red diamond dekat.

   Kekurangan:

   - Tidak efisien jika red diamond terlalu jauh → membuang waktu.

2. **Greedy Berdasarkan Jarak Terdekat (Greedy by Nearest Distance)**

   Bot akan selalu memilih objek (diamond) yang paling dekat dari posisi saat ini.

   Prioritas:

   - Mengutamakan kecepatan pengambilan objek (waktu tempuh minimum).
   - Tidak mempertimbangkan apakah diamond merah atau biru.

   Kelebihan:

   - Cocok untuk permainan cepat atau sisa waktu sedikit.
   - Menghindari pemborosan langkah.

   Kekurangan:

   - Bisa mengumpulkan banyak blue diamond, tapi total poin tetap rendah.

3. **Greedy Berdasarkan Density (Greedy by Value per Distance)**

   Bot memilih objek berdasarkan rasio nilai terhadap jarak:

   $$\text{Distance} = \frac{Poin\ Objek}{Jarak\ dari\ Bot\ ke\ Objek}$$

   Prioritas:

   - Objek dengan rasio poin-per-jarak tertinggi akan dipilih.

   Kelebihan:

   - Seimbang antara keuntungan (poin) dan efisiensi langkah.
   - Mampu menghasilkan skor tinggi dalam waktu terbatas.

   Kekurangan:

   - Perlu perhitungan lebih rumit dibanding dua pendekatan sebelumnya.

4. **Greedy dengan Perhitungan Sisa Waktu (Greedy by Time-Aware Selection)**

   Mirip dengan pendekatan density, tapi mempertimbangkan juga sisa waktu dan kebutuhan kembali ke base.

   Prioritas:

   - Memilih objek dengan nilai tinggi yang masih bisa dijangkau dan sempat kembali ke base.

Kelebihan:

- Menghindari penalti karena gagal kembali ke base.
- Adaptif terhadap waktu tersisa.

Kekurangan:

- Kompleks dan memerlukan pemantauan waktu secara terus-menerus.

5. **Greedy Kombinasi dengan Prioritas Khusus**

   Menggabungkan beberapa strategi, dengan pengaturan bobot atau kondisi. Misalnya:

   - Saat waktu masih panjang → pakai density.
   - Saat waktu hampir habis → pakai nearest.
   - Jika red button terlihat → prioritaskan.

   Kelebihan:

   - Lebih adaptif terhadap situasi permainan.
   - Bisa dioptimalkan dengan kondisi map tertentu.

   Kekurangan:

   - Lebih kompleks dalam pengaturan logika.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

1. **Greedy Berdasarkan Nilai Maksimal**

   Strategi ini efektif hanya jika objek dengan nilai tinggi (red diamond) berada dalam jarak yang wajar atau dapat dijangkau dengan cepat. Namun, ketika red diamond berada jauh dari posisi bot, strategi ini akan membuat bot menghabiskan waktu terlalu lama untuk mengejar satu objek, sehingga menurunkan efisiensi total poin yang dikumpulkan.

   Cocok Digunakan Saat:

   - Map kecil dan red diamond tersebar merata.
   - Tidak banyak rintangan atau musuh.

Tidak Efektif Jika:

- Red diamond terlalu jauh atau terhalang rintangan.
- Waktu terbatas.

2. Greedy Berdasarkan Jarak Terdekat

Strategi ini menjamin bahwa bot akan selalu bergerak dengan cepat ke objek terdekat, memaksimalkan penggunaan waktu. Namun, karena tidak mempertimbangkan nilai dari objek, maka bisa terjadi bot hanya mengambil banyak blue diamond (1 poin), sehingga total skor rendah meskipun objek yang dikumpulkan banyak.

Cocok Digunakan Saat:

- Waktu permainan sangat terbatas.
- Bot perlu menghindari kerumitan rintangan atau musuh.

Tidak Efektif Jika:

- Banyak red diamond yang seharusnya bisa diambil dengan sedikit usaha lebih.

3. Greedy Distance

Strategi ini dianggap paling seimbang dan umumnya menghasilkan skor yang optimal. Dengan membandingkan nilai terhadap jarak, bot akan mengejar objek yang memberikan rasio keuntungan terbaik. Strategi ini sangat cocok untuk permainan dengan durasi menengah hingga panjang, di mana efisiensi tiap langkah sangat berpengaruh terhadap skor total.

Cocok Digunakan Saat:

- Waktu cukup panjang.
- Bot perlu menyeimbangkan antara poin dan kecepatan gerak.

Tidak Efektif Jika:

- Waktu sangat terbatas, tidak sempat menghitung rasio terlalu kompleks.

4. Greedy dengan Perhitungan Sisa Waktu

Strategi ini mengadaptasi pendekatan density tetapi memasukkan waktu tersisa sebagai faktor utama. Sangat cocok untuk mendekati akhir permainan, agar bot tidak terjebak

terlalu jauh dari base dan kehilangan seluruh poin karena tidak sempat kembali. Meski kompleks, strategi ini lebih realistis dalam skenario kompetitif.

Cocok Digunakan Saat:

- Sisa waktu mendekati habis.
- Bot berada jauh dari base dan harus segera kembali.

Tidak Efektif Jika:

- Waktu permainan masih sangat panjang (berpotensi terlalu defensif).

5. **Greedy Kombinasi**

   Pendekatan kombinasi merupakan bentuk strategi adaptif yang menyesuaikan logika greedy tergantung situasi. Strategi ini menggabungkan kelebihan semua strategi sebelumnya, misalnya:

   - Awal permainan → pakai density.
   - Menjelang akhir → perhitungan waktu.
   - Jika red button atau teleporter terlihat → prioritas spesifik.

   Namun, strategi ini lebih sulit diimplementasikan karena membutuhkan banyak kondisi, logika percabangan, dan pengambilan keputusan dinamis.

   Cocok Digunakan Saat:

   - Ingin memaksimalkan skor secara konsisten dalam berbagai kondisi map.
   - Kompetisi yang melibatkan berbagai macam situasi dan peta dinamis.

   Tidak Efektif Jika:

   - Bot masih sederhana dan belum cukup fleksibel dalam logika pengambilan keputusan.

Berdasarkan analisis tersebut, strategi greedy distance menjadi pilihan paling efektif untuk implementasi awal karena mampu menyeimbangkan antara keuntungan poin dan jarak secara efisien, dengan kompleksitas yang masih cukup terjangkau. Strategi kombinasi bisa dikembangkan di tahap lanjutan saat bot sudah cukup matang.

3.4 Strategi Greedy yang Dipilih

Dari berbagai alternatif strategi greedy yang telah dipikirkan dan dianalisis sebelumnya, strategi yang akhirnya kami pilih untuk diimplementasikan dalam game Diamonds adalah Greedy berdasarkan nilai per jarak (value/distance) atau sering disebut juga Greedy distance.

Strategi yang akhirnya kami pilih untuk diimplementasikan dalam permainan Diamonds adalah strategi Greedy berdasarkan nilai per jarak (Greedy distance). Strategi ini dipilih karena menawarkan pendekatan yang seimbang antara nilai keuntungan dan efisiensi waktu/langkah. Dalam konteks permainan, bot perlu memaksimalkan poin sebanyak mungkin dalam waktu yang terbatas. Dengan menggunakan rasio:

$$\text{Distance} = \frac{Poin\ Objek}{Jarak\ dari\ Bot\ ke\ Objek}$$

bot dapat menentukan objek mana yang paling "menguntungkan" untuk dikejar pada setiap langkah. Ini memungkinkan bot untuk:

- Tidak hanya fokus pada poin tinggi (seperti red diamond),
- Tapi juga tetap menghemat waktu tempuh,
- Dan menghindari objek jauh yang kurang efisien.

Kelebihan Strategi Ini

- Cocok untuk berbagai jenis peta dan skenario permainan.
- Tetap memperhitungkan efisiensi pergerakan.
- Lebih realistis dibanding hanya mengejar objek terdekat atau objek dengan nilai tertinggi saja.
- Memudahkan pengembangan lanjutan, misalnya dikombinasikan dengan strategi perhitungan sisa waktu di tahap selanjutnya.

Pada setiap langkah:

1. Bot menghitung jarak (menggunakan Manhattan Distance) ke seluruh objek yang tersedia.
2. Menghitung nilai density dari setiap objek (poin dibagi jarak).
3. Memilih objek dengan density tertinggi yang masih feasible (misal: inventory belum penuh untuk red diamond).

4. Bergerak menuju objek tersebut.

Dengan strategi ini, diharapkan bot dapat mengambil keputusan yang lebih cerdas dan efisien, sehingga mampu mengumpulkan poin secara optimal dan tetap bisa kembali ke base tepat waktu.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

Algoritma Greedy Distance

game > logic > bot_greedy_distance.py > ...

```python
import argparse

from game.models import Game, Board, GameObject

# Import bot-botmu
from bot_greedy_distance import GreedyDistanceBot
from bot_greedy_value import GreedyValueBot
from bot_greedy_value_distance import GreedyValueDistanceBot

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('--bot', choices=['distance', 'value', 'value_distance'], default='distance', help='Choose bot to run')
    args = parser.parse_args()

    if args.bot == 'distance':
        bot = GreedyDistanceBot()
    elif args.bot == 'value':
        bot = GreedyValueBot()
    elif args.bot == 'value_distance':
        bot = GreedyValueDistanceBot()
    else:
        raise ValueError("Unknown bot")

    # Inisialisasi game, board, dll (sesuaikan dengan kode asli kamu)
    game = Game(bot)
    game.run()

if __name__ == '__main__':
    main()

```

Algoritma Greedy Value

game > logic > bot_greedy_value.py > ...

```python
from typing import Optional, List

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

def indexValid(x, y: int, width, height):
    return 0 <= x < width and 0 <= y < height

class GreedyValueBot(BaseLogic):
    def __init__(self):
        self.directions = [(1,0), (0,1), (-1,0), (0,-1)]
        self.bot = None
        self.board_width = 0
        self.board_height = 0
        self.diamonds: List[GameObject] = []
        self.redButton = None

    def distance(self, pos1: Position, pos2: Position) -> int:
        return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

    def get_direction_towards(self, curr: Position, dest: Position) -> (int, int):
        dx = clamp(dest.x - curr.x, -1, 1)
        dy = clamp(dest.y - curr.y, -1, 1)
        if dx != 0:
            return dx, 0
        else:
            return 0, dy

    def next_move(self, board_bot: GameObject, board: Board):
        self.bot = board_bot
        self.board_width = board.width
        self.board_height = board.height
        self.diamonds = []
        self.redButton = None

        for obj in board.game_objects:
            if obj.type == "DiamondGameObject":
                self.diamonds.append(obj)
            elif obj.type == "DiamondButtonGameObject":
                self.redButton = obj
```

```
42
43          time_left = board_bot.properties.milliseconds_left // 1000
44          dist_to_base = self.distance(board_bot.position, board_bot.properties.base)
45          if board_bot.properties.diamonds >= board_bot.properties.inventory_size or time_left <= dist_to_base + 1:
46              if not self.positions_equal(board_bot.position, board_bot.properties.base):
47                  return self.get_direction_towards(board_bot.position, board_bot.properties.base)
48              else:
49                  return 0, 0
50
51          max_point = -1
52          candidate_diamonds = []
53          for diamond in self.diamonds:
54              if board_bot.properties.diamonds + diamond.properties.points <= board_bot.properties.inventory_size:
55                  if diamond.properties.points > max_point:
56                      max_point = diamond.properties.points
57                      candidate_diamonds = [diamond]
58                  elif diamond.properties.points == max_point:
59                      candidate_diamonds.append(diamond)
60
61          if not candidate_diamonds:
62              if not self.positions_equal(board_bot.position, board_bot.properties.base):
63                  return self.get_direction_towards(board_bot.position, board_bot.properties.base)
64              else:
65                  return 0, 0
66
67          # Pilih yang terdekat di antara diamond dengan poin tertinggi
68          min_dist = float('inf')
69          target = None
70          for diamond in candidate_diamonds:
71              dist = self.distance(board_bot.position, diamond.position)
72              if dist < min_dist:
73                  min_dist = dist
74                  target = diamond
75
76          return self.get_direction_towards(board_bot.position, target.position)
77
78      def positions_equal(self, pos1: Position, pos2: Position) -> bool:
79          return pos1.x == pos2.x and pos1.y == pos2.y
80
```

Algoritma Greedy Value Distance

game > logic > bot_greedy_value_distance.py > GreedyValueDistanceBot > __init__

```python
from typing import Optional, List

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

def indexValid(x, y: int, width, height):
    return 0 <= x < width and 0 <= y < height

class GreedyValueDistanceBot(BaseLogic):
    def __init__(self):
        self.directions = [(1,0), (0,1), (-1,0), (0,-1)]
        self.bot = None
        self.board_width = 0
        self.board_height = 0
        self.diamonds: List[GameObject] = []
        self.redButton = None

    def distance(self, pos1: Position, pos2: Position) -> int:
        return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

    def get_direction_towards(self, curr: Position, dest: Position) -> (int, int):
        dx = clamp(dest.x - curr.x, -1, 1)
        dy = clamp(dest.y - curr.y, -1, 1)
        if dx != 0:
            return dx, 0
        else:
            return 0, dy

    def next_move(self, board_bot: GameObject, board: Board):
        self.bot = board_bot
        self.board_width = board.width
        self.board_height = board.height
        self.diamonds = []
        self.redButton = None
```

```python
        for obj in board.game_objects:
            if obj.type == "DiamondGameObject":
                self.diamonds.append(obj)
            elif obj.type == "DiamondButtonGameObject":
                self.redButton = obj

        time_left = board_bot.properties.milliseconds_left // 1000
        dist_to_base = self.distance(board_bot.position, board_bot.properties.base)
        if board_bot.properties.diamonds >= board_bot.properties.inventory_size or time_left <= dist_to_base + 1:
            if not self.positions_equal(board_bot.position, board_bot.properties.base):
                return self.get_direction_towards(board_bot.position, board_bot.properties.base)
            else:
                return 0, 0

        best_score = -1
        target = None
        for diamond in self.diamonds:
            if board_bot.properties.diamonds + diamond.properties.points <= board_bot.properties.inventory_size:
                dist = self.distance(board_bot.position, diamond.position)
                score = diamond.properties.points / (dist + 1)
                if score > best_score:
                    best_score = score
                    target = diamond

        if target is None:
            if not self.positions_equal(board_bot.position, board_bot.properties.base):
                return self.get_direction_towards(board_bot.position, board_bot.properties.base)
            else:
                return 0, 0

        return self.get_direction_towards(board_bot.position, target.position)

    def positions_equal(self, pos1: Position, pos2: Position) -> bool:
        return pos1.x == pos2.x and pos1.y == pos2.y

```

2. Penjelasan Alur Program

- **Algoritma Greedy Distance (GreedyDistanceBot)**
  Bot ini memilih gerakan berdasarkan jarak terdekat ke objek target (misalnya diamond) dengan cara mencari posisi terdekat dari bot saat ini ke objek yang ingin dikumpulkan. Bot akan selalu bergerak ke arah yang mengurangi jarak Manhattan (jarak absolut di sumbu x dan y) menuju target tersebut. Jika bot sudah membawa cukup banyak item (misalnya diamond) atau waktu tersisa semakin sedikit, bot akan kembali ke posisi dasar (base).
- **Algoritma Greedy Value (GreedyValueBot)**
  Bot ini memilih target berdasarkan nilai poin tertinggi dari objek (diamond) yang tersedia, bukan hanya jarak terdekat. Pertama, bot mencari diamond dengan nilai poin terbesar yang masih muat di inventori. Jika ada lebih dari satu diamond dengan nilai poin tertinggi yang sama, bot memilih yang paling dekat di antara mereka. Seperti sebelumnya, jika inventori penuh atau waktu hampir habis, bot kembali ke base.
- **Algoritma Greedy Value Distance (GreedyValueDistanceBot)**
  Algoritma ini merupakan gabungan dari dua pendekatan di atas dengan memaksimalkan rasio antara nilai poin diamond dengan jarak ke diamond

tersebut. Dengan kata lain, bot menghitung skor sebagai nilai poin / (jarak + 1) dan memilih target dengan skor tertinggi. Pendekatan ini memungkinkan bot memilih target dengan nilai tinggi tapi juga mempertimbangkan jarak agar tidak membuang waktu terlalu lama.

4.2 Struktur Data yang Digunakan

Program ini menggunakan beberapa struktur data yang telah tersedia di dalam game dan beberapa variabel yang memanfaatkan struktur data tersebut. Berikut adalah rincian struktur data dan variabel yang dipakai dalam program.

1. Position

   class Position:
   x: int
   y: int

   Kelas ini menyimpan informasi tentang posisi dalam permainan, yang terdiri dari koordinat x dan y pada papan permainan (board).
2. Properties

   class Properties:
   points: int = None
   pair_id: str = None
   diamonds: int = None
   score: int = None
   name: str = None
   inventory_size: int = None
   can_tackle: bool = None
   milliseconds_left: int = None
   time_joined: str = None
   base: Position = None

   Kelas ini menyimpan properti tambahan dari sebuah objek dalam game. Properti ini meliputi poin objek (misalnya nilai diamond), ID pasangan, jumlah diamond yang dimiliki, skor, nama, kapasitas inventori, kemampuan tackle, waktu tersisa dalam milidetik, waktu bergabung, dan posisi base dari bot.
3. GameObject

   class GameObject:
   id: int
   position: Position
   type: str
   properties: Properties = None

   Kelas ini merepresentasikan objek dalam permainan, seperti bot, diamond, tombol, atau base. Atributnya meliputi id unik, posisi objek yang berupa instance dari Position, tipe

objek (misalnya "DiamondGameObject" atau "Bot"), serta properti tambahan yang dikemas dalam instance kelas Properties.

4. Variabel yang Digunakan

   Berikut adalah beberapa variabel penting dalam program beserta fungsinya:

- **bot** : GameObject yang merepresentasikan bot yang sedang dikendalikan oleh algoritma.
- **board_width, board_height** : integer yang menyimpan ukuran papan permainan.
- **diamonds** : list dari GameObject yang berisi semua diamond yang ada di papan saat ini.
- **redButton** : GameObject yang menyimpan informasi tombol merah (jika ada).
- **time_left** : integer waktu tersisa untuk bot dalam permainan (dalam detik).
- **dist_to_base** : integer jarak antara posisi bot dengan base.
- **best_score** / **max_point** : nilai numerik yang digunakan untuk memilih target diamond terbaik berdasarkan nilai tertentu (skor atau poin).
- **target** : GameObject yang merupakan diamond yang dipilih sebagai tujuan gerak selanjutnya.

4.3 Pengujian Program

1. Skenario Pengujian

Pengujian dilakukan dengan beberapa skenario untuk mengamati perilaku ketiga algoritma bot dalam kondisi yang berbeda. Pada skenario pertama, diamond tersebar merata di papan permainan dengan waktu yang cukup banyak tersedia. Ketiga bot diuji untuk memilih diamond mana yang akan diambil. Bot Greedy Distance lebih memilih diamond yang paling dekat dari posisinya saat ini tanpa mempertimbangkan nilai diamond tersebut. Sebaliknya, bot Greedy Value memilih diamond dengan poin tertinggi, tanpa terlalu memperhatikan jarak. Sedangkan bot Greedy Value Distance menyeimbangkan antara nilai poin dan jarak, dengan mempertimbangkan rasio antara keduanya untuk menentukan target yang optimal.

Pada skenario kedua, bot hampir penuh dengan diamond di inventori dan waktu yang tersisa mulai mendekati habis. Dalam kondisi ini, ketiga bot diharapkan segera kembali ke base untuk menyimpan hasil koleksi mereka. Hal ini penting agar poin yang sudah dikumpulkan tidak hilang akibat kehabisan waktu. Skenario ketiga menguji kasus ketika terdapat beberapa diamond dengan nilai yang sama namun berada di posisi berbeda. Di sini, bot Greedy Value akan memilih diamond dengan nilai yang sama namun akan memprioritaskan yang paling dekat agar efisien dalam pergerakan. Skenario terakhir menguji kondisi ketika tidak ada diamond tersisa di papan, di mana bot seharusnya tetap diam atau langsung kembali ke base karena tidak ada lagi objek yang bisa dikumpulkan.

2. Hasil Pengujian dan Analisis

Hasil pengujian menunjukkan bahwa bot Greedy Distance efektif pada situasi di mana jarak ke target merupakan prioritas utama. Namun, bot ini kurang optimal dalam mengumpulkan diamond bernilai tinggi yang berada agak jauh, sehingga total poin yang diperoleh bisa lebih rendah dibanding bot lain. Bot Greedy Value cenderung mengabaikan jarak dan memilih diamond dengan nilai terbesar, sehingga bot

bisa menghabiskan waktu lebih banyak berjalan menuju diamond yang jauh, meningkatkan risiko kehabisan waktu. Meski begitu, jika manajemen waktu dapat dilakukan dengan baik, bot ini mampu mendapatkan poin maksimal dari diamond bernilai besar. Sementara itu, bot Greedy Value Distance memberikan keseimbangan terbaik antara nilai poin dan efisiensi jarak. Dengan menggabungkan nilai dan jarak dalam perhitungan skor, bot ini mampu mengoptimalkan total poin yang dapat dikumpulkan dalam waktu yang tersedia. Pada skenario ketika waktu hampir habis dan inventori penuh, ketiga bot secara konsisten kembali ke base, yang menandakan fungsi pengelolaan waktu dan inventori berjalan dengan baik. Meski performa ketiga algoritma dapat berbeda tergantung pada layout papan dan distribusi diamond, secara umum Greedy Value Distance menunjukkan adaptasi yang lebih baik dan hasil yang lebih optimal.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil implementasi dan pengujian ketiga algoritma greedy pada bot permainan Diamonds, dapat disimpulkan bahwa setiap algoritma memiliki keunggulan dan keterbatasan yang bergantung pada kondisi permainan. Algoritma **Greedy Distance** efektif saat prioritas utama adalah efisiensi jarak dengan memilih target terdekat, namun kurang optimal dalam mengambil diamond bernilai tinggi yang letaknya jauh. Algoritma **Greedy Value** memprioritaskan pengambilan diamond dengan nilai tertinggi, sehingga mampu mengejar objek bernilai besar meski berisiko menghabiskan waktu lebih banyak akibat jarak tempuh yang panjang. Sedangkan algoritma **Greedy Value Distance** yang menggabungkan nilai dan jarak dalam perhitungan rasio poin per jarak menunjukkan performa paling adaptif dan konsisten, menyeimbangkan efisiensi gerak dan kualitas target sehingga mengoptimalkan hasil akhir bot. Ketiga algoritma juga mampu mengelola waktu dan inventori dengan baik, ditandai dengan tindakan kembali ke base saat inventory penuh atau waktu hampir habis, menjaga poin yang sudah terkumpul tidak hilang. Namun, efektivitas algoritma greedy sangat bergantung pada distribusi diamond, posisi musuh, dan dinamika waktu, karena sifat greedy yang fokus pada keputusan lokal tanpa perencanaan jangka panjang. Implementasi ini menunjukkan bagaimana strategi greedy dapat diterapkan secara praktis dengan dukungan struktur data yang tepat dan pengelolaan sumber daya. Secara keseluruhan, Greedy Value Distance merupakan pendekatan paling optimal dalam konteks permainan ini, walaupun untuk performa yang lebih baik dan tahan terhadap situasi kompleks diperlukan pengembangan lebih lanjut yang mengintegrasikan prediksi, evaluasi risiko, dan strategi multi-langkah adaptif.

5.2 Saran

Meskipun bot yang dikembangkan menggunakan strategi greedy telah mampu menjalankan tugas dengan baik dalam mengumpulkan diamond secara efisien, masih terdapat beberapa aspek yang dapat dikembangkan lebih lanjut agar bot menjadi lebih cerdas dan kompetitif. Beberapa saran pengembangan tersebut antara lain:

1. **Pertimbangan Dinamis terhadap Musuh**

   Saat ini bot belum memperhitungkan keberadaan musuh secara aktif. Ke depannya, strategi dapat dikembangkan agar bot dapat:

   - Menghindari jalur yang dilalui musuh.
   - Mengatur waktu pengambilan diamond agar tidak berebut dengan bot lain.
   - Menghitung risiko jika musuh lebih dekat ke target yang sama.

2. **Optimalisasi dengan Algoritma Lain (Hybrid Approach)**

   Menggabungkan algoritma greedy dengan algoritma lain seperti:

   - **A*** untuk pencarian jalur optimal ke diamond atau base.
   - **Dynamic Programming** untuk merencanakan rute dengan banyak target.
   - **Minimax atau Expectimax** jika permainan dibuat lebih kompetitif.

3. **Manajemen Waktu dan Inventori yang Lebih Cermat**

   Bot dapat dikembangkan untuk:

   - Lebih bijak dalam menentukan kapan harus kembali ke base.
   - Memperhitungkan kapasitas inventori dan menyesuaikan target diamond berdasarkan sisa ruang.

4. **Pemanfaatan Red Button dan Teleporter secara Strategis**

   Red Button dan teleporter merupakan elemen permainan yang dapat dimanfaatkan lebih optimal. Pengembangan dapat dilakukan agar:

   - Bot memicu Red Button pada waktu yang menguntungkan.
   - Bot menggunakan teleporter untuk mempercepat pergerakan atau kabur dari musuh.

5. **Penyesuaian Strategi Berdasarkan Fase Permainan**

   Strategi greedy bisa dibuat adaptif terhadap waktu. Contohnya:

   - Di awal permainan, fokus pada diamond dengan density tinggi.
   - Di pertengahan permainan, mengatur pergerakan untuk menghindari konflik.
   - Di akhir permainan, prioritas pada penyimpanan diamond ke base.

Dengan mengimplementasikan saran-saran di atas, diharapkan performa bot akan meningkat, tidak hanya dari segi efisiensi pengambilan diamond, tetapi juga dari kemampuan adaptasi terhadap situasi permainan yang kompleks dan dinamis.

LAMPIRAN

A. Repository Github

https://github.com/willysyifa/Tubes1_Botyy.git

B. Video Penjelasan (link GDrive)

Link Drive:

https://drive.google.com/file/d/1w7JMw713sT0dRYk83blGVLaN12d1D1oW/view?usp=sharing

Link Youtube: https://youtu.be/E-bmih_hZjQ?feature=shared

DAFTAR PUSTAKA

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

[2] A. V. Aho, J. E. Hopcroft, dan J. D. Ullman, *Data Structures and Algorithms*. Reading, MA: Addison-Wesley, 1983.

[3] J. Kleinberg dan É. Tardos, *Algorithm Design*. Boston, MA: Pearson/Addison-Wesley, 2005.

[4] GeeksforGeeks, “Greedy Algorithm,” [Online]. Tersedia: https://www.geeksforgeeks.org/greedy-algorithms/. [Diakses: 1-Jun-2025].

[5] Visualgo, “Greedy Strategy Visualization,” [Online]. Tersedia: https://visualgo.net/en/dfsbfs. [Diakses: 1-Jun-2025].

[6] IF2211 Strategi Algoritma, “Dokumentasi Permainan Diamonds,” Institut Teknologi Bandung, 2025.