

IMPLEMENTASI ALGORITMA GREEDY GACORBOT DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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



Oleh: Kelompok 9 (9ame 9acor)

Ardiansyah Fernando 123140102

Hezkiel Rajani Aritonang 123140118

Andryano Shevchenko Limbong 123140205

Dosen Pengampu: Winda Yulita, M.Cs.

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

DAFTAR ISI

BAB I	DESKRIPSI TUGAS.....	3
BAB II	LANDASAN TEORI.....	4
2.1	Dasar Teori.....	4
1.	Cara Implementasi Program.....	6
2.	Menjalankan Bot Program.....	7
3.	Langkah-langkah Bot Berjalan.....	8
BAB III	APLIKASI STRATEGI GREEDY.....	10
3.1	Proses Mapping.....	10
3.2	Eksplorasi Alternatif Solusi Greedy.....	14
3.3	Analisis Efisiensi dan Efektivitas Solusi Greedy.....	15
3.4	Strategi Greedy yang Dipilih.....	16
BAB IV	IMPLEMENTASI DAN PENGUJIAN.....	21
4.1	Implementasi Algoritma Greedy.....	21
1.	Pseudocode.....	21
2.	Penjelasan Alur Program.....	26
4.2	Struktur Data yang Digunakan.....	32
4.3	Pengujian Program.....	35
4.3.1.	Skenario Pengujian.....	35
4.3.2.	Hasil Pengujian dan Analisis.....	41
BAB V	KESIMPULAN DAN SARAN.....	43
5.1	Kesimpulan.....	43
5.2	Saran.....	43
LAMPIRAN.....		45
DAFTAR PUSTAKA.....		46

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.

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

BAB II

LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy adalah jenis algoritma yang memilih tindakan paling optimal pada setiap langkahnya, dengan harapan akan menghasilkan solusi yang paling optimal, efektif, efisien, dan benar. [1].

Prinsip utama dari Algoritma Greedy adalah ambil yang terbaik yang bisa didapat sekarang sehingga algoritma akan memilih opsi yang paling menguntungkan, dengan harapan keputusan tersebut akan mengarahkan ke opsi-opsi yang lebih menguntungkan dan optimal secara keseluruhan. Namun, penting untuk diingat bahwa keputusan yang telah diambil tidak akan diubah di langkah selanjutnya, sehingga algoritma ini tidak mempertimbangkan kemungkinan revisi keputusan di masa depan.

1. Ciri-ciri algoritma greedy adalah sebagai berikut:

- Solusi dibangun secara bertahap: Solusi akhir adalah hasil dari serangkaian keputusan-keputusan pada tahap sebelumnya yang diambil.
- Tidak ada revisi keputusan: Setelah sebuah keputusan diambil, algoritma greedy tidak dapat lagi mengubahnya.
- Optimal Substructure: Masalah yang diselesaikan memiliki struktur di mana solusi optimal dari submasalah digunakan untuk membentuk solusi optimal dari masalah utama.
- Greedy Choice Property: Pilihan terbaik pada setiap langkah akan mengarah pada solusi optimal secara keseluruhan.

2. Algoritma Greedy memiliki beberapa komponen utama, sebagai berikut:

- Himpunan Kandidat (C): Kumpulan elemen yang dapat dipilih untuk membentuk solusi optimal.
- Himpunan Solusi (S): Kumpulan elemen yang telah dipilih dan membentuk solusi sementara.
- Fungsi Seleksi: Mekanisme untuk memilih kandidat yang terbaik yang tersedia.
- Fungsi Kelayakan: Fungsi yang menentukan kandidat dipilih atau tidak sehingga tidak melanggar batasan masalah.
- Fungsi Objektif: Fungsi yang digunakan untuk mengevaluasi seberapa baik solusi yang telah dipilih.

3. Kelebihan Algoritma Greedy adalah sebagai berikut:

- Efisiensi Waktu: Algoritma greedy biasanya memiliki kompleksitas waktu yang lebih rendah dibandingkan metode algoritma yang lain.
- Implementasi Sederhana: Pendekatan algoritma greedy yang secara langsung memudahkan implementasinya.

4. Kekurangan Algoritma Greedy adalah sebagai berikut:

- Tidak selalu menghasilkan solusi optimal: Algoritma greedy tidak selalu menghasilkan solusi optimal secara keseluruhan, terutama jika masalah tidak memenuhi property *greedy choice* dan *optimal substructure*.

- Keputusan Irreversible: Keputusan yang sudah buat, tidak ada mekanisme untuk revisi dan mengubah keputusan tersebut sehingga menjadi masalah jika keputusan awal ternyata tidak optimal.

2.2 Cara Kerja Program

Secara umum, cara kerja program bot beroperasi sebagai agen otonom yang secara terus-menerus menerima data kondisi terkini game engine melalui API, kemudian menganalisis informasi tersebut menggunakan logika internalnya yang sudah terprogram untuk mengambil keputusan, dan setiap keputusan atau langkah yang diambil akan menghasilkan opsi yang paling menguntungkan secara lokal berdasarkan kriteria, diamond dengan nilai tertinggi dan jarak paling pendek.

1. Cara Implementasi Program

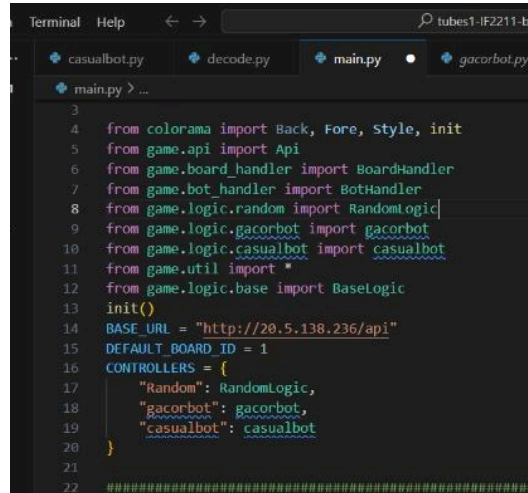
- Membuat Folder.

Langkah pertama dari implementasi algoritma greedy pada bot ialah dengan membuat folder baru pada direktori /game/logic (misalnya mybot.py).

- Membuat Class Program.

Langkah kedua adalah membuat class program yang dapat melakukan pewarisan class BaseLogic, lalu implementasikan constructor dan method next_move.

- Import Class Program pada main.py untuk didaftarkan pada dictionary.



```

3
4 from colorama import Back, Fore, Style, init
5 from game.api import Api
6 from game.board_handler import BoardHandler
7 from game.bot_handler import BotHandler
8 from game.logic.random import RandomLogic
9 from game.logic.gacorbob import gacorbob
10 from game.logic.casualbot import casualbot
11 from game.util import *
12 from game.logic.base import BaseLogic
13 init()
14 BASE_URL = "http://20.5.138.236/api"
15 DEFAULT_BOARD_ID = 1
16 CONTROLLERS = {
17     "Random": RandomLogic,
18     "gacorbob": gacorbob,
19     "casualbot": casualbot
20 }
21
22

```

- Jalankan Program.

Jalankan program seperti step c pada bagian 2, sesuaikan argumen logic pada command/script menjadi nama bot yang telah terdaftar pada controllers. Dapat menjalankan 1 bit atau beberapa bot menggunakan .bat atau .sh script.

2. Menjalankan Bot Program

- Environments:

Game Engine Diamonds memerlukan 3 program aplikasi, yaitu, Node.js, Docker desktop, dan Yarn yang dapat didownload melalui website yang telah disediakan dan diinstalasi sekaligus konfigurasi sesuai dengan petunjuk yang telah disediakan pada tutorial.

- Menjalankan Game Engine.

Game Engine harus dijalankan terlebih dahulu di localhost. Engine akan terkoneksi dengan program bot yang telah dibuat.

- Menjalankan Bot.

Bot dapat dijalankan dengan code program yang dibuat dengan menggunakan bahasa pemrograman python. Kemudian bot bergerak sesuai dengan algoritma greedy yang telah dibuat.

3. Langkah-langkah Bot Berjalan

- Menerima Data Permainan Menggunakan API.

Bot diprogram untuk menerima informasi kondisi permainan secara langsung melalui API (Application Programming Interface) yang telah disediakan oleh program permainan “Diamonds”. Data-datanya, ialah sebagai berikut:

- o Posisi bot (x, y) dan status inventory (jumlah diamond yang telah diambil dan disimpan)
- o Posisi dan jenis dari diamond, baik merah atau biru yang ada di atas papan permainan diamonds.
- o Posisi base dari bot, tempat diamond merah atau biru akan diantarkan.
- o Posisi 2 teleporter, yang saling terhubung satu sama lain.
- o Posisi Red Button.
- o Skor dan sisa waktu permainan.

- Analisis Oleh Bot

Berdasarkan data yang diterima oleh bot menggunakan API. Bot kemudian mengolah data-data tersebut menggunakan logic algoritma greedy yang telah diprogram sehingga menghasilkan himpunan kandidat atau aksi yang valid.

- Pengambilan Keputusan.

Setelah analisis dilakukan sehingga menghasilkan himpunan kandidat, program bot pun akan mengambil keputusan sesuai dengan algoritma greedy, yaitu, memilih aksi yang mengarah ke diamond dengan poin paling besar dan jarak yang paling dekat.

- Mengirim Perintah Aksi ke Game Engine.

Setelah keputusan diambil, program bot akan mengirimkan perintah aksi untuk dapat bergerak ke arah diamond yang diinginkan, yaitu, “NORTH” untuk ke atas bot, “SOUTH” untuk ke arah bawah bot, “WEST” untuk ke arah kiri bot dan “EAST” untuk ke arah kanan bot.

- Update Status Permainan Secara Otomatis

Game Engine akan memproses aksi dan pergerakan dari bot dan memperbarui kondisi permainan secara real time.

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

- Himpunan Kandidat (C)

Setiap giliran bot bergerak, himpunan kandidat terdiri dari semua kemungkinan langkah yang dapat dipilih oleh bot untuk bergerak, yaitu, sebagai berikut:

- o Bergerak ke salah satu sel tetangga (NORTH “UTARA”, SOUTH “SELATAN”, WEST “BARAT”, EAST ”TIMUR”).
- o Base: tempat untuk menyimpan diamond yang dibawa.
- o Diamond Biru Terdekat dipilih berdasarkan jarak antara grid.
- o Diamond Merah Terdekat dipilih berdasarkan jarak antara grid, prioritasnya lebih tinggi jika berada di dalam jarak yang sama dengan diamond biru.
- o Bergerak menuju teleporter terdekat jika teleporter lebih efisien seperti, teleporter lebih dekat menuju ke base dibandingkan berjalan per grid.
- o Bergerak menuju red button jika diamond telah berjumlah sedikit dan butuh di reset.
- o Bergerak menuju base jika inventory telah penuh.

- Himpunan Solusi.

Himpunan solusi adalah kumpulan elemen yang dipilih secara bertahap berdasarkan keputusan optimal pada pada setiap langkah, tanpa mempertimbangkan dampak pilihan tersebut terhadap langkah berikutnya, akan tetapi tetap diharapkan akan membentuk solusi akhir yang optimal atau paling tidak mendekati hasil yang

optimal. [2]. Akan tetapi, pada program logika bot akan berfokus pada solusi-solusi yang diambil secara aksi per aksi

- Fungsi Solusi.

Permainan diamonds akan selesai apabila waktu telah habis. Fungsi Solusi akhir adalah total skor yang berhasil dikumpulkan oleh bot. Fungsi solusi akan terpenuhi apabila bot berhasil mengumpulkan diamond biru atau merah dari papan permainan dan menyimpannya ke base.

Fungsi solusi akan tercapai ketika waktu permainan sudah berakhir, keberhasilan solusi diukur dari total skor yang dikumpulkan oleh program bot.

- Fungsi Seleksi.

Fungsi seleksi adalah inti dari strategi algoritma greedy. Fungsi seleksi berfungsi untuk memilih aksi terbaik dari himpunan kandidat pada setiap langkah.

- o Logika Kondisi Mendesak Kembali Ke Base:

1. Jika jarak ke base kritis relatif terhadap sisa waktu.
2. Jika jarak ke base sangat dekat (1 atau 2 langkah blok grid) dan membawa lebih dari 2 diamond (atau lebih dari 0 diamond jika jarak 1).
3. Jika inventory penuh (membawa 5 diamond).

```
def next_move(self, board_bot: GameObject, board: Board):
    gcor = board_bot.properties
    posisi_saat_ini = board_bot.position
    base = gcor.base

    if self.jarakbase(board_bot) in {gcor.milliseconds_left, 2} and gcor.diamonds > 2 or \
        self.jarakbase(board_bot) == 1 and gcor.diamonds > 0 or gcor.diamonds == 5:
        self.goal_position = base
```

o Logika Mengumpulkan Diamond (jika gcor.diamonds ≥ 3):

1. Diamond merah terdekat jika jaraknya ≤ 3 .
2. Jika tidak, diamond biru terdekat jika jarak ≤ 3 .
3. Jika tidak, diamond terdekat di sekitar base (jika ada).
4. Jika tidak, kembali ke base.

```
elif gcor.diamonds >= 3:
    if self.diamond_terdekat(board_bot, board) is not None or self.diamondmerah_terdekat(board_bot, board) is not None:
        if gcor.diamonds == 3 and self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
            self.goal_position = self.diamondmerah_terdekat(board_bot, board)
        elif self.jarak_diamond_dekat(board_bot, board) <= 3:
            self.goal_position = self.diamond_terdekat(board_bot, board)
        else:
            base = board_bot.properties.base
            self.goal_position = base
    elif self.diamondsekitarbase(board_bot, board):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    else:
        base = board_bot.properties.base
        self.goal_position = base
```

o Logika Mengumpulkan Diamond (jika gcor.diamonds < 3):

1. Diamond terdekat di sekitar base, jika bot juga berada di sekitar base atau jika ada ≥ 3 diamond di sekitar base.
2. Jika tidak, memprioritaskan diamond merah terdekat, mempertimbangkan jarak jika ada diamond biru juga.
3. Jika tidak ada merah, mengambil diamond biru terdekat.
4. Jika tidak ada diamond biru terdekat, dan tombol merah lebih dekat daripada diamond biru terdekat, memilih tombol merah.
5. Jika tidak, kembali ke base.

```

elif gcor.diamonds < 3:
    if (self.diamondsekitarbase(board_bot, board) and self.botsekitarbase(board_bot)) or (
        self.diamondsekitarbase(board_bot, board) and len(self.diamond_dekat_base(board_bot, board)) >= 3
    ):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    elif self.diamondmerah_terdekat(board_bot, board) is not None:
        if self.diamond_terdekat(board_bot, board) is not None:
            if self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
                self.goal_position = self.diamondmerah_terdekat(board_bot, board)
            else:
                self.goal_position = self.diamond_terdekat(board_bot, board)
        else:
            self.goal_position = self.diamondmerah_terdekat(board_bot, board)
    elif self.diamond_terdekat(board_bot, board) is not None:
        self.goal_position = self.diamond_terdekat(board_bot, board)

    elif self.jarak_diamond_tmblmrh(board_bot, board):
        self.goal_position = self.caritmblmrh(board).position

    else:
        self.goal_position = board_bot.properties.base

```

o Penggunaan Teleporter

Jika tujuan adalah base dan belum dalam mode teleportasi, bot akan mengevaluasi penggunaan teleporter terdekat jika teleporter mempersingkat jarak ke base dan memenuhi kondisi jarak.

```

if self.goal_position is None:
    self.goal_position = base

if self.goal_position == base and not self.is_teleport:
    self.teleport_ke_base(board_bot, board)

delta_x, delta_y = self.peroleh_jarak(
    posisi_saat_ini.x, posisi_saat_ini.y,
    self.goal_position.x, self.goal_position.y
)

return delta_x, delta_y

```

- Fungsi Kelayakan

Fungsi Kelayakan adalah fungsi yang memeriksa apakah elemen yang telah dipilih dapat dimasukkan ke dalam himpunan solusi tanpa melanggar batasan yang diberikan. [3]

Pergerakan yang dilakukan oleh program bot dianggap layak, jika:

1. Target tersebut dipilih berdasarkan aturan prioritas dalam method `next_move`.
2. Kondisi jarak untuk mengambil diamond terpenuhi.

- Fungsi Objektif

Fungsi objektif memiliki fungsi untuk mengevaluasi kualitas solusi untuk memaksimalkan skor yang didapat serta meminimalkan pergerakan yang dilakukan. [3]

Fungsi objektif yang ingin dimaksimalkan oleh program bot yang dibuat adalah skor akhir yang diperoleh dengan mengumpulkan diamond dan menyimpannya di base dengan pergerakan seminimal mungkin.

3.2 Eksplorasi Alternatif Solusi Greedy

Pada algoritma greedy terdapat beberapa pendekatan yang mungkin digunakan sebagai alternatif atau dasar pengembangan:

- Greedy Berdasarkan Jarak Terdekat Murni
- Greedy Berdasarkan Nilai Diamond Tertinggi.
- Greedy Dengan Skor Per Langkah (SPLE).
- Greedy dengan Manajemen Inventory Sederhana.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

- Greedy Berdasarkan Jarak Terdekat Murni

Bot akan selalu bergerak ke diamond terdekat tanpa memandang jenis atau kondisi inventory. Strategi ini sederhana namun seringkali tidak efektif dan efisien karena mengabaikan nilai diamond.

- Greedy Berdasarkan Nilai Diamond Tertinggi.

Bot akan selalu memprioritaskan diamond merah dengan nilai paling tinggi dibandingkan diamond biru. Hal ini dapat menghasilkan skor tinggi per diamond tetapi mungkin tidak efisien, apabila posisi diamond merah jauh dari posisi bot berada.

- Greedy Dengan Skor Per Langkah (SPLE).

Program logika bot akan menghitung rasio poin dan jarak untuk setiap diamond dan memilih dengan rasio skor tertinggi sehingga menghasilkan skor dan pergerakan yang seimbang. Alternatif ini merupakan salah satu solusi yang mungkin paling efektif dan efisien

- Greedy dengan Manajemen Inventory Sederhana.

Program bot akan mengatur isi inventory untuk mencapai ambang batas (dalam sistem ini memiliki batas maksimal 5) sebelum kembali ke base. Sistem ini tidak efektif dan efisien karena bot tidak mempertimbangkan jarak dan fokus memenuhi inventory.

3.4 Strategi Greedy yang Dipilih

Program logika bot milik kelompok kami tidak menerapkan satupun pendekatan alternatif algoritma greedy di atas secara murni. Akan tetapi, program logika bot milik kami mengimplementasikan strategi greedy berbasis aturan atau rule base greedy strategy yang kompleks dengan menggabungkan solusi alternatif yang berbeda satu sama lain.

Kelompok kami memilih untuk menggabungkan keempat solusi alternatif tersebut dengan beberapa kondisi tambahan melalui fungsi `next_move` sehingga menghasilkan strategi algoritma greedy yang baik dan dapat mengevaluasi kondisi permainan saat itu. Berikut adalah rincian alur logika yang kami buat:

- Prioritas untuk Kembali ke Base untuk Mengamankan Skor.
 - Pemicu:
 1. Jika jarak ke base sama dengan sisa milidetik atau jarak ke base adalah 2 langkah, dan bot sedang membawa lebih dari 2 diamond.
 2. Atau, jika jarak ke base adalah 1 langkah dan bot membawa diamond lebih dari 0.
 3. Atau, jika bot membawa 5 diamond atau kapasitas inventory penuh.
 - Keputusan: `self.goal_position = base` bot bergerak ke base.

```
if self.jarakbase(board_bot) in {gcor.milliseconds_left, 2} and gcor.diamonds > 2 or \
    self.jarakbase(board_bot) == 1 and gcor.diamonds > 0 or gcor.diamonds == 5:
    self.goal_position = base
```

Alasan karena di dalam kondisi ini, pilihan greedy yang paling logis adalah mengamankan diamond ke dalam base untuk mendapatkan skor.

- Prioritas untuk Mengumpulkan Diamond Saat Inventory Cukup Terisi (`gcor.diamonds ≥ 3`).
 - Pemicu:
 1. Bot membawa 3 atau 4 diamond.

- Sub-prioritas (pilihan greedy lokal bertingkat):
 1. Target Diamond Merah Dekat: Jika ada diamond merah terdekat dengan jarak ≤ 3 , self.goal_position adalah diamond merah tersebut.
 2. Target Diamond Biru Dekat: Jika tidak, dan ada diamond biru terdekat dengan jarak ≤ 3 , self.goal_position adalah diamond biru tersebut.
 3. Target Diamond Strategis di Sekitar Base: Jika tidak ada diamond dekat (merah/biru) yang memenuhi syarat, dan ada diamond di sekitar base (dalam radius 4 dari base seperti didefinisikan diamond_dekat_base), self.goal_position adalah diamond terdekat dari kumpulan tersebut (menggunakan diamonddekatbot).
 4. Kembali ke Base: Jika tidak ada target diamond di atas, self.goal_position adalah base.

```

elif gcor.diamonds >= 3:
    if self.diamond_terdekat(board_bot, board) is not None or self.diamondmerah_terdekat(board_bot, board) is not None:
        if gcor.diamonds == 3 and self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
            self.goal_position = self.diamondmerah_terdekat(board_bot, board)
        elif self.jarak_diamond_dekat(board_bot, board) <= 3:
            self.goal_position = self.diamond_terdekat(board_bot, board)
        else:
            base = board_bot.properties.base
            self.goal_position = base
    elif self.diamondsekitarbase(board_bot, board):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    else:
        base = board_bot.properties.base
        self.goal_position = base

```

- Prioritas untuk Mengumpulkan Diamond Saat Inventory SedikitTerisi (gcor.diamonds < 3).
 - Pemicu:
 1. Bot membawa diamond kurang dari 3.
 - Sub-prioritas (pilihan greedy lokal bertingkat):
 1. Target Diamond Strategis di Sekitar Base: Jika ada diamond di sekitar base (radius 2, via diamondsekitarbase) DAN (bot juga berada di dekat base (radius 4, via botsekitarbase) ATAU jumlah diamond di sekitar base (radius 4) ≥ 3), self.goal_position adalah diamond terdekat dari kumpulan

tersebut (menggunakan `diamonddekatbot` dari `diamond_dekat_base`).
(Rakus memanfaatkan area produktif dekat base).

2. Target Diamond Merah (Prioritas Nilai): Jika (1) tidak terpenuhi dan ada diamond merah:
 - a. Jika ada juga diamond biru, dan diamond merah berjarak ≤ 3 , targetkan diamond merah.
 - b. Jika tidak (merah jauh atau tidak ada biru yang relevan untuk perbandingan ini), targetkan diamond biru terdekat (jika ada).
 - c. Jika hanya ada diamond merah (tidak ada biru), targetkan diamond merah. (Strategi ini memprioritaskan diamond merah jika sangat dekat, jika tidak, ia mungkin beralih ke biru atau merah tergantung keberadaan dan jarak).
3. Target Diamond Biru (Pilihan Dasar): Jika (1) dan (2) tidak terpenuhi, dan ada diamond biru, targetkan diamond biru terdekat.
4. Target Tombol Merah: Jika (1-3) tidak terpenuhi, dan tombol merah lebih dekat daripada diamond biru terdekat (berdasarkan fungsi `jarak_diamond_tmblmrh`), targetkan posisi tombol merah.
5. Kembali ke Base (Fallback): Jika tidak ada target lain, `self.goal_position` adalah base.

```
elif gcor.diamonds < 3:
    if (self.diamondsekitarbase(board_bot, board) and self.botsekitarbase(board_bot)) or (
        self.diamondsekitarbase(board_bot, board) and len(self.diamond_dekat_base(board_bot, board)) >= 3
    ):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    elif self.diamondmerah_terdekat(board_bot, board) is not None:
        if self.diamond_terdekat(board_bot, board) is not None:
            if self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
                self.goal_position = self.diamondmerah_terdekat(board_bot, board)
            else:
                self.goal_position = self.diamond_terdekat(board_bot, board)
        else:
            self.goal_position = self.diamondmerah_terdekat(board_bot, board)
    elif self.diamond_terdekat(board_bot, board) is not None:
        self.goal_position = self.diamond_terdekat(board_bot, board)

    elif self.jarak_diamond_tmblmrh(board_bot, board):
        self.goal_position = self.caritmbmlrh(board).position

    else:
        self.goal_position = board_bot.properties.base
```

- Prioritas Kembali ke Base.
 - Pemicu: Jika `self.goal_position` masih `None` setelah semua evaluasi di atas.
 - Keputusan: `self.goal_position` ditetapkan ke `base`.

```
if self.goal_position is None:  
    self.goal_position = base
```

Alasan, untuk memastikan bot selalu memiliki tujuan dan kembali ke base adalah default yang aman.

- Prioritas Menggunakan Teleporter Untuk Kembali ke Base
 - Pemicu: Jika `self.goal_position` adalah `base` dan bot belum dalam proses teleportasi (`self.is_teleport == False`).
 - Keputusan: Fungsi `teleport_ke_base` dievaluasi. Jika ditemukan rute via teleporter yang lebih pendek ke base (dengan syarat jarak bot ke teleporter masuk ≤ 5 dan total jarak gabungan lebih pendek dari jarak langsung ke base), `self.goal_position` diubah ke posisi teleporter masuk, dan `self.is_teleport` diaktifkan.

```
if self.goal_position == base and not self.is_teleport:  
    self.teleport_ke_base(board_bot, board)
```

Alasan, bot dapat berpindah secara cepat ke base untuk menyimpan diamond.

- Menentukan Gerakan (`peroleh_jarak`)
 - Fungsi `peroleh_jarak` memilih gerakan (`delta_x`, `delta_y`) yang paling mengurangi jarak pada sumbu (`x` atau `y`) yang memiliki selisih absolut terbesar.

```
delta_x, delta_y = self.peroleh_jarak(  
    posisi_saat_ini.x, posisi_saat_ini.y,  
    self.goal_position.x, self.goal_position.y  
)  
  
return delta_x, delta_y
```

Alasan, program bot dapat mengambil langkah yang paling signifikan untuk mengurangi jarak ke target pada satu sumbu.

Walaupun strategi-strategi gacor bot di atas terlihat rumit. Akan tetapi program logika bot yang kami buat tetap menggunakan strategi algoritma greedy sehingga setiap keputusan lokal terbaik akan diambil tanpa melakukan perencanaan secara global yang mendalam.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
from typing import Optional
from typing import List
from game.logic.base import BaseLogic
from game.models import Board, GameObject, Position

class gacorbob(BaseLogic):
    def __init__(self):
        self.arah = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] = None
        self.is_teleport = False
        self.langkah = 0
        self.arah_saat_ini = 0

    def diamond_dekat_base(self, bot_papan: GameObject, papan: Board, jarak: int = 4):
        gcor = bot_papan.properties.base
        diamond = papan.diamonds
        if not gcor:
            return []

        return [
            diamond.position for diamond in diamond
            if ((gcor.x - jarak <= diamond.position.x <= gcor.x + jarak) and
                (gcor.y - jarak <= diamond.position.y <= gcor.y + jarak))
        ]

    def botsekitarbase(self, bot_papan: GameObject, jarak: int = 4):
        gcor = bot_papan.properties.base
        posisi_saat_ini = bot_papan.position
        return ((gcor.x - jarak <= posisi_saat_ini.x <= gcor.x + jarak) and
                (gcor.y - jarak <= posisi_saat_ini.y <= gcor.y + jarak))

    def diamonddekatbot(self, bot_papan: GameObject, diamonds: List[Position]):
```

```

posisi_saat_ini = bot_papan.position
    diamond_terdekat = min(diamonds, key=lambda diamond: abs(diamond.x - posisi_saat_ini.x) +
abs(diamond.y - posisi_saat_ini.y))
    return diamond_terdekat

def diamondsekitarbase(self, bot_papan: GameObject, papan: Board, jarak: int = 2):
    diamonds = papan.diamonds
    gcor = bot_papan.properties.base

    if not gcor:
        return False

    for diamond in diamonds:
        if ((gcor.x - jarak) <= diamond.position.x <= (gcor.x + jarak) and
            (gcor.y - jarak) <= diamond.position.y <= (gcor.y + jarak)):
            return True

    return False

def diamond_terdekat(self, bot_papan: GameObject, papan: Board):
    posisi_saat_ini = bot_papan.position
    diamond_biru = [d for d in papan.diamonds if d.properties.points == 1]
    if not diamond_biru:
        return None

    return min(diamond_biru, key=lambda d: abs(d.position.x - posisi_saat_ini.x) + abs(d.position.y
- posisi_saat_ini.y)).position if diamond_biru else None

def jarak_diamond_dekat(self, bot_papan: GameObject, papan: Board):
    terdekat = self.diamond_terdekat(bot_papan, papan)
    return 999 if terdekat is None else abs(terdekat.x - bot_papan.position.x) + abs(terdekat.y -
bot_papan.position.y)

def diamondmerah_terdekat(self, bot_papan: GameObject, papan: Board):
    posisi_saat_ini = bot_papan.position
    diamond_merah = [d for d in papan.diamonds if d.properties.points == 2]
    if not diamond_merah:
        return None

```

```

        return min(diamond_merah, key=lambda d: abs(d.position.x - posisi_saat_ini.x) +
abs(d.position.y - posisi_saat_ini.y)).position if diamond_merah else None

def jarak_diamondmerah_dekat(self, bot_papan: GameObject, papan: Board):
    terdekat = self.diamondmerah_terdekat(bot_papan, papan)
    return 999 if terdekat is None else abs(terdekat.x - bot_papan.position.x) + abs(terdekat.y -
bot_papan.position.y)

def jarakbase(self, bot_papan: GameObject):
    base, pos = bot_papan.properties.base, bot_papan.position
    return abs(base.x - pos.x) + abs(base.y - pos.y)

def hitungjarak(self, pos1, pos2):
    return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

def caribotlain(self, bot_papan: GameObject, papan: Board):
    return [bot for bot in papan.bots if bot.id != bot_papan.id
        and bot.properties.base != bot.position
        and bot.properties.diamonds >= 3
        and bot.properties.diamonds > bot_papan.properties.diamonds]

def kejar_bot_musuh(self, bot_papan: GameObject, papan: Board):
    if self.jarakbase(bot_papan) > 4 or self.langkah > 5:
        self.goal_position, self.langkah, self.is_teleport = None, 0, False
    return False

for bot in self.caribotlain(bot_papan, papan):
    dist = self.hitungjarak(bot_papan.position, bot.position)
    if dist == 0:
        self.goal_position = bot_papan.properties.base
        return False
    elif dist <= 3:
        self.goal_position = bot.position
        return True

self.goal_position = None
return False

def caritmbmlmrh(self, papan: Board):

```

```

        return next((item for item in papan.game_objects if item.type == "DiamondButtonGameObject"),
None)

    def hitungjaraktmbmlrh(self, bot_papan: GameObject, papan: Board):
        tmbmlrh = self.caritmbmlrh(papan)
        return (abs(tmbmlrh.position.x - bot_papan.position.x) + abs(tmbmlrh.position.y -
bot_papan.position.y)) if tmbmlrh else float('inf')

    def jarak_diamond_tmbmlrh(self, bot_papan: GameObject, papan: Board):
        tmbmlrh = self.caritmbmlrh(papan)

        if not tmbmlrh:
            return False

        diamond_biru_terdekat = self.diamond_terdekat(bot_papan, papan)
        if not diamond_biru_terdekat:
            return False

        return self.hitungjaraktmbmlrh(bot_papan, papan) < self.jarak_diamond_dekat(bot_papan,
papan)

    def cariteleporter(self, bot_papan: GameObject, papan: Board):
        teleporters = [item for item in papan.game_objects if item.type == "TeleportGameObject"]
        return sorted(teleporters, key=lambda tele: self.hitungjarak(tele.position, bot_papan.position))

    def teleport_ke_base(self, bot_papan: GameObject, papan: Board):
        teleporters = self.cariteleporter(bot_papan, papan)

        if len(teleporters) < 2:
            return

        teleporter_dekat = teleporters[0]
        teleporter_baik = min(teleporters, key=lambda tele: self.hitungjarak(tele.position,
bot_papan.properties.base))

        jarakkeBase = self.hitungjarak(teleporter_baik.position, bot_papan.properties.base)
        jarakkebot = self.hitungjarak(bot_papan.position, teleporter_dekat.position)

        if jarakkebot <= 5 and jarakkeBase + jarakkebot < self.jarakbase(bot_papan):

```



```

self.is_teleport= True
self.goal_position = teleporter_dekat.position

def peroleh_jarak(self, current_x, current_y, dest_x, dest_y):
    x = -1 if dest_x < current_x else 1
    y = -1 if dest_y < current_y else 1

    dx, dy = (x, 0) if abs(dest_x - current_x) >= abs(dest_y - current_y) else (0, y)
    return dx, dy

def next_move(self, board_bot: GameObject, board: Board):
    gcor = board_bot.properties
    posisi_saat_ini = board_bot.position
    base = gcor.base

    if self.jarakbase(board_bot) in {gcor.milliseconds_left, 2} and gcor.diamonds > 2 or \
        self.jarakbase(board_bot) == 1 and gcor.diamonds > 0 or gcor.diamonds == 5:
        self.goal_position = base

    elif gcor.diamonds >= 3:
        if self.diamond_terdekat(board_bot, board) is not None or
self.diamondmerah_terdekat(board_bot, board) is not None:
            if gcor.diamonds == 3 and self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
                self.goal_position = self.diamondmerah_terdekat(board_bot, board)
            elif self.jarak_diamond_dekat(board_bot, board) <= 3:
                self.goal_position = self.diamond_terdekat(board_bot, board)
            else:
                base = board_bot.properties.base
                self.goal_position = base
        elif self.diamondsekitarbase(board_bot, board):
            diamond_list = self.diamond_dekat_base(board_bot, board)
            self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
        else:
            base = board_bot.properties.base
            self.goal_position = base

    elif gcor.diamonds < 3:
        if (self.diamondsekitarbase(board_bot, board) and self.botsekitarbase(board_bot)) or (

```

```

        self.diamondsekitarbase(board_bot, board) and len(self.diamond_dekat_base(board_bot,
board))) >= 3
    ):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    elif self.diamondmerah_terdekat(board_bot, board) is not None:
        if self.diamond_terdekat(board_bot, board) is not None:
            if self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
                self.goal_position = self.diamondmerah_terdekat(board_bot, board)
            else:
                self.goal_position = self.diamond_terdekat(board_bot, board)
        else:
            self.goal_position = self.diamondmerah_terdekat(board_bot, board)
    elif self.diamond_terdekat(board_bot, board) is not None:
        self.goal_position = self.diamond_terdekat(board_bot, board)

    elif self.jarak_diamond_tmblmrh(board_bot, board):
        self.goal_position = self.caritmbmlrh(board).position

    else:
        self.goal_position = board_bot.properties.base

    if self.goal_position is None:
        self.goal_position = base

    if self.goal_position == base and not self.is_teleport:
        self.teleport_ke_base(board_bot, board)

    delta_x, delta_y = self.peroleh_jarak(
        posisi_saat_ini.x, posisi_saat_ini.y,
        self.goal_position.x, self.goal_position.y
    )

    return delta_x, delta_y

```

2. Penjelasan Alur Program

Di dalam konstruktor gacorbot terdapat 5 atribut:

- arah yang akan mengatur arah gerak seperti barat/kiri(-1, 0), timur/kanan(0,1), selatan/bawah(0, -1), dan utara/atas(0,1)
- goal_position yang mengatur tujuan dari bot, karena belum memiliki tujuan maka bernilai None
- is_teleport boolean yang mengecek apakah bot sedang menggunakan teleporter untuk teleport
- langkah yang menghitung jumlah langkah yang ditempuh bot
- arah_saat_ini yang mengatur arah pergerakan bot saat bergerak

Ada 19 fungsi/method di dalam program yang kami buat:

- diamond_dekat_base yang akan mengatur fungsi mencari lokasi diamond di dekat base dalam radius 4 grid ke atas, bawah, kiri, kanan dan akan mengembalikan nilai jika terdapat diamond di dalam radius pencarian tersebut. Jika base tidak ada maka mengembalikan list kosong.
- botsekitarbase yang akan menghitung apakah bot dalam radius sekitar base, kami set radius yang sama seperti sebelumnya yaitu 4 ke atas, bawah, kiri, dan kanan.
- diamonddekatbot yang akan mengembalikan nilai jarak diamond_terdekat yang berada di dekat bot, menggunakan min untuk mencari jarak terkecil dari hasil perhitungan jarak bot dengan diamond menggunakan rumus perhitungan manhattan. key=lambda digunakan untuk mendefinisikan fungsi anonim, dalam konteks ini adalah min.
- diamondsekitarbase yang akan mengecek apakah ada diamond di sekitar base, berbeda dengan fungsi diamond_dekat_base yang akan mengembalikan nilai jarak diamond di dekat base, fungsi ini mengecek apakah memang ada sedikitnya 1 diamond di sekitar base. Jika ada maka akan mengembalikan nilai True, jika tidak maka akan mengembalikan nilai False.
- diamond_terdekat dan diamondmerah_terdekat memiliki mekanisme yang hampir sama, perbedaannya adalah properties.value, dimana diamond bernilai 1 (diamond biru) dan diamondmerah bernilai 2. Pencarian dilakukan dengan mencari nilai terkecil menggunakan min dari hasil perhitungan jarak manhattan lalu mengembalikan posisi dari diamond dan diamondmerah pada papan. Kode juga

mengecek apakah ada `diamond_biru` dan `diamond_merah`, jika kosong akan mengembalikan nilai `None`.

- `jarak_diamond_dekat` dan `jarak_diamondmerah_dekat` untuk menghitung jarak `diamond` dan `diamondmerah` terdekat dengan bot dari hasil pencarian yang dilakukan fungsi `diamond_terdekat` dan `diamondmerah_terdekat` menggunakan rumus manhattan. Jika `diamond` tidak dalam jarak jangkauan bot, maka mengembalikan nilai 999, jika masih bisa dijangkau menggunakan jarak perhitungan menggunakan rumus manhattan berdasarkan posisi `diamond` dan posisi bot pada papan.
- `jarakbase` digunakan untuk menghitung jarak bot di papan ke base menggunakan perhitungan rumus manhattan lalu nilai perhitungan akan dikembalikan ke bot.
- `hitungjarak` digunakan untuk menghitung jarak dari posisi awal ke posisi tujuan menggunakan rumus manhattan dan akan mengembalikan nilai ke bot. Fungsi ini akan digunakan dalam fungsi selanjutnya dalam mencari posisi objek seperti bot lawan, teleporter, dan tombol merah.
- `caribotlain` digunakan untuk mengembalikan daftar bot yang memenuhi syarat, dalam fungsi kami menulis jika bukan bot yang kami buat yaitu `gacorbot`, tidak berada di base, memiliki sedikitnya 3 diamonds dalam inventory, dan memiliki jumlah `diamond` lebih banyak dari bot kami maka akan mengembalikan daftar botnya yang memenuhi.
- `kejar_bot_musuh` merupakan fungsi yang tidak akan mengejar bot jika jarak bot ke base sendiri lebih dari 4 dan langkah yang diperlukan lebih dari 5. Menggunakan perulangan untuk mengecek setiap bot dari hasil daftar `caribotlain` kemudian jika jarak bot ke bot musuh adalah 0 atau posisi sama maka akan kembali ke base, jika jarak maksimal bot dan bot musuh adalah 3 maka akan melakukan pengejaran. Jika tidak ada musuh, maka bot tidak akan set bot musuh sebagai `goal_position`.
- `caritmbmlmrh` merupakan fungsi untuk mencari objek pertama yang ditemukan di papan yaitu objek tombol merah sesuai nama di game.

- `hitungjaraktmbmlrh` merupakan fungsi yang menghitung jarak bot ke tombol merah yang telah ditemukan sebelumnya menggunakan rumus manhattan, jika tidak ada tombol merah maka akan mengembalikan jarak tak berhingga.
- `jarak_diamond_tmbmlrh` merupakan fungsi untuk membandingkan nilai jarak bot ke tombol merah dan nilai bot ke diamond biru. Jika tidak ada tombol merah dan diamond yang ditemukan maka akan mengembalikan nilai `False`.
- `cariteleporter` akan mengembalikan nilai jarak yang telah diurutkan terkait posisi teleporter yang dekat dengan bot, mekanisme pencarian sama seperti tombol merah yang mencari sesuai nama item dalam property base game. Perhitungan nilai jarak menggunakan rumus manhattan seperti biasanya.
- `teleport_ke_base` merupakan fungsi yang akan menggunakan teleporter jika jarak bot ke teleporter + jarak teleporter tujuan ke base lebih dekat dibandingkan dengan berjalan per grid. Jika jumlah teleporter pada papan kurang dari 2 maka fungsi tidak akan melanjutkan ke perhitungan jarak dan efisiensi pilihan apakah menggunakan teleporter atau tidak.
- `peroleh_jarak` merupakan fungsi yang menentukan arah pergerakan bot sesuai dengan arah tujuan ke atas, bawah, kiri atau kanan
- `next_move`, fungsi ini dibagi menjadi 3 bagian yang sebaiknya kami jelaskan satu persatu:

```
if self.jarakbase(board_bot) in {gcor.milliseconds_left, 2} and gcor.diamonds > 2 or \
    self.jarakbase(board_bot) == 1 and gcor.diamonds > 0 or gcor.diamonds == 5:
    self.goal_position = base
```

Ini akan set `goal_position` ke base, jika jarak bot ke base dan memiliki diamond lebih dari 2 serta waktu tersisa adalah 2 detik, atau jarak bot ke base adalah 1 dan diamond yang dimiliki adalah 1 artinya disamping base terdapat diamond biru, atau inventory gacorbob telah penuh yaitu memiliki 5 diamonds maka akan kembali ke base.

```

elif gcor.diamonds >= 3:
    if self.diamond_terdekat(board_bot, board) is not None or
self.diamondmerah_terdekat(board_bot, board) is not None:
        if gcor.diamonds == 3 and self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
            self.goal_position = self.diamondmerah_terdekat(board_bot, board)
        elif self.jarak_diamond_dekat(board_bot, board) <= 3:
            self.goal_position = self.diamond_terdekat(board_bot, board)
        else:
            base = board_bot.properties.base
            self.goal_position = base
    elif self.diamondsekitarbase(board_bot, board):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    else:
        base = board_bot.properties.base
        self.goal_position = base

```

Akan mengecek apakah ada diamond merah dan diamond biru dekat. Lalu jika diamond di dalam inventory tepat ada 3 maka akan mencari diamond merah terdekat dan jika jarak ke diamond merah maksimal 3 langkah. Jika ada diamond_biru dan posisi maksimal ke diamond biru adalah 3 maka bot akan ke diamond biru. Jika jarak bot ke diamond lebih dari 3 maka akan kembali ke base, dan akan scan di sekitar base, jika ada diamond di dekat base maka akan mengambil diamond tersebut sebelum kembali ke base. Jika tidak ada diamond juga di dekat base maka akan langsung kembali ke base.

```

elif gcor.diamonds < 3:
    if (self.diamondsekitarbase(board_bot, board) and self.botsekitarbase(board_bot)) or (
        self.diamondsekitarbase(board_bot, board) and
len(self.diamond_dekat_base(board_bot, board)) >= 3
    ):
        diamond_list = self.diamond_dekat_base(board_bot, board)
        self.goal_position = self.diamonddekatbot(board_bot, diamond_list)
    elif self.diamondmerah_terdekat(board_bot, board) is not None:
        if self.diamond_terdekat(board_bot, board) is not None:
            if self.jarak_diamondmerah_dekat(board_bot, board) <= 3:
                self.goal_position = self.diamondmerah_terdekat(board_bot, board)
            else:

```

```

        self.goal_position = self.diamond_terdekat(board_bot, board)
    else:
        self.goal_position = self.diamondmerah_terdekat(board_bot, board)
    elif self.diamond_terdekat(board_bot, board) is not None:
        self.goal_position = self.diamond_terdekat(board_bot, board)

    elif self.jarak_diamond_tmblmrh(board_bot, board):
        self.goal_position = self.caritmblmrh(board).position

    else:
        self.goal_position = board_bot.properties.base

```

Jika diamond yang dimiliki kurang dari 3 maka bot akan mencari di sekitar base untuk menghemat waktu, jika ditemukan ada diamond dekat base maka bot akan mengambil diamond tersebut. Jika ada diamond merah yang dekat dengan bot dengan jarak maksimal 3 maka akan ke arah diamond merah, jika tidak dekat maka akan menuju ke diamond biru yang dekat dengan bot. Jika ada tombol merah yang lebih dekat dibandingkan diamond maka bot justru akan menuju ke tombol merah. Jika semua itu tidak terpenuhi, tidak ada diamond biru ataupun diamond merah yang dekat dengan base atau bot, atau jika tidak ada tombol merah yang dekat maka akan kembali ke base.

Jika semua percabangan telah dicek dan dijalankan maka akan menjalankan percabangan sisa di akhir next_move

Jika bot tidak memiliki goal position atau goal position = None maka akan kembali ke base

Jika bot ingin kembali ke base dan belum teleport jika menemukan teleport yang lebih efektif digunakan maka bot akan menggunakan teleporter

Terakhir delta_x dan delta_y akan menggunakan fungsi peroleh_jarak untuk menentukan arah pergerakan berdasarkan posisi bot ke tujuan bot. delta_x dan delta_y akan mengembalikan nilai perubahan posisi yang dilakukan oleh bot dalam papan.

4.2 Struktur Data yang Digunakan

Pada program gacorbot, kami menggunakan beberapa struktur data utama untuk menyimpan dan mengelola informasi terkait posisi objek, arah pergerakan, serta status bot selama permainan. Berikut penjelasannya:

1. List Arah

- Digunakan untuk menentukan empat kemungkinan arah pergerakan bot dalam bentuk tuple (x, y).

Contoh :

```
Python
self.arah = [(1, 0), (0, 1), (-1, 0), (0, -1)]
```

Keterangan:

(1, 0) → ke kanan

(0, 1) → ke bawah

(-1, 0) → ke kiri

(0, -1) → ke atas

2. Position

- Merupakan struktur data bawaan engine untuk menyimpan koordinat X dan Y dari berbagai objek di board.

contoh :

```
Python
posisi_saat_ini = board_bot.position
```


3. List Diamond dan Bot lain

- Diamond dan bot lain diambil dalam bentuk list dari Board.

Contoh :

```
Python
diamonds = board.diamonds

bots = board.bots
```

4. Properti Bot

- Berisi informasi status bot saat ini seperti jumlah diamond yang dibawa, sisa waktu permainan, dan posisi base.

Contoh :

```
Python
gcor = board_bot.properties

base = gcor.base
```

5. Variabel Goal Position

- Menyimpan tujuan pergerakan bot dalam bentuk Position.

Contoh :

```
Python
self.goal_position = Position(x, y)
```

6. Flag Boolean

- Digunakan untuk status kondisi seperti teleportasi.

Contoh :

```
Python  
self.is_teleport = False
```

7. Variabel Integer

- Untuk menghitung jumlah langkah pengejaran musuh.

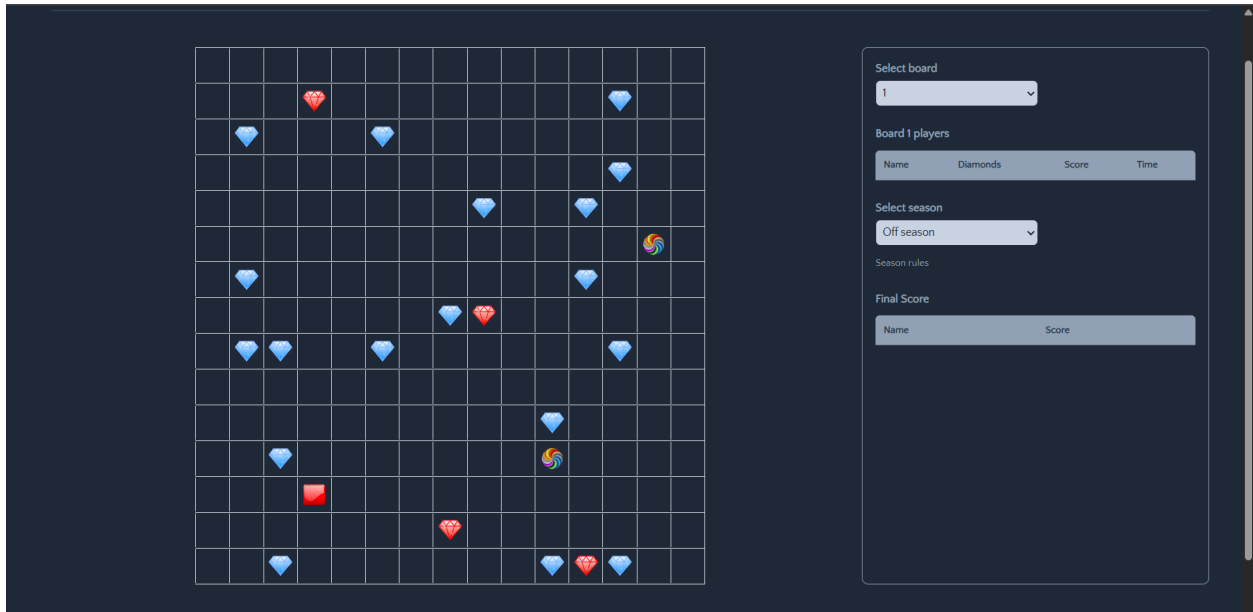
Contoh :

```
Python  
self.langkah = 0
```

4.3 Pengujian Program

4.3.1. Skenario Pengujian

1. Pengujian Jumlah pengambilan diamond (gacorbot)



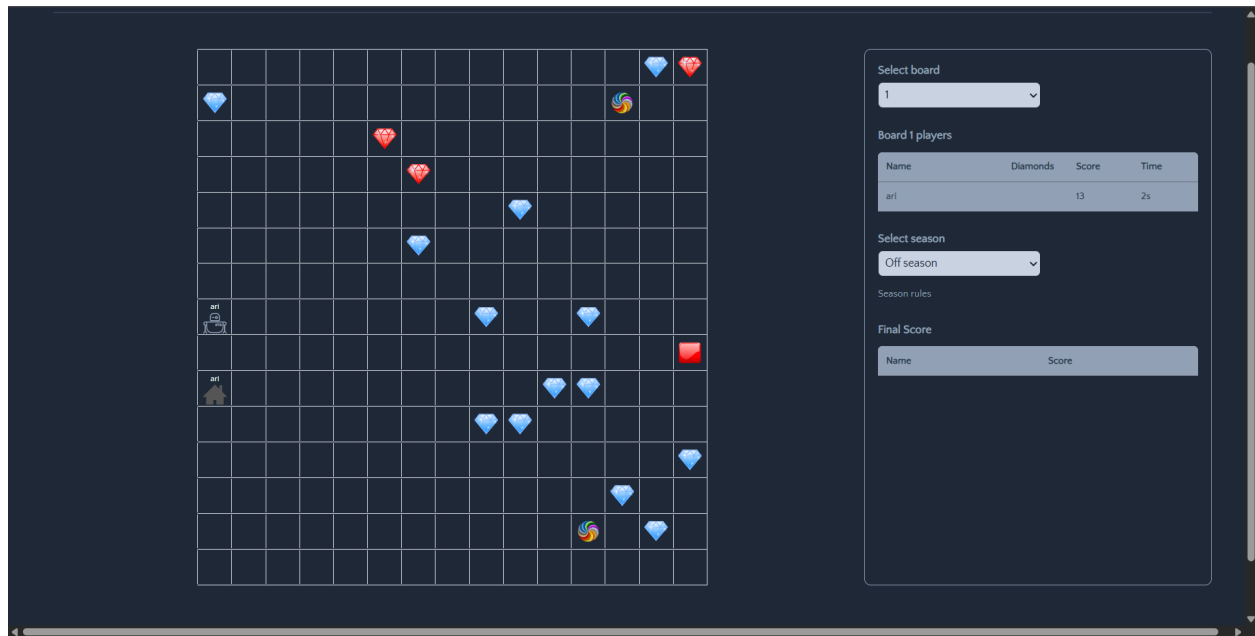
gambaran awal map (sebelum bot dimasukkan)

```
Microsoft Windows [Version 10.0.26100.4202]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Hezkiel>cd tubes1-IF2211-bot-starter-pack-1.0.1

C:\Users\Hezkiel\tubes1-IF2211-bot-starter-pack-1.0.1>python main.py --logic gacorbot --email=kiel@gmail.com --name=ari
--password=anjay --team etimo
```

aktifkan bot dengan cmd (python main.py --logic gacorbot --email=kiel@gmail.com --name=ari
--password=anjay --team etimo)

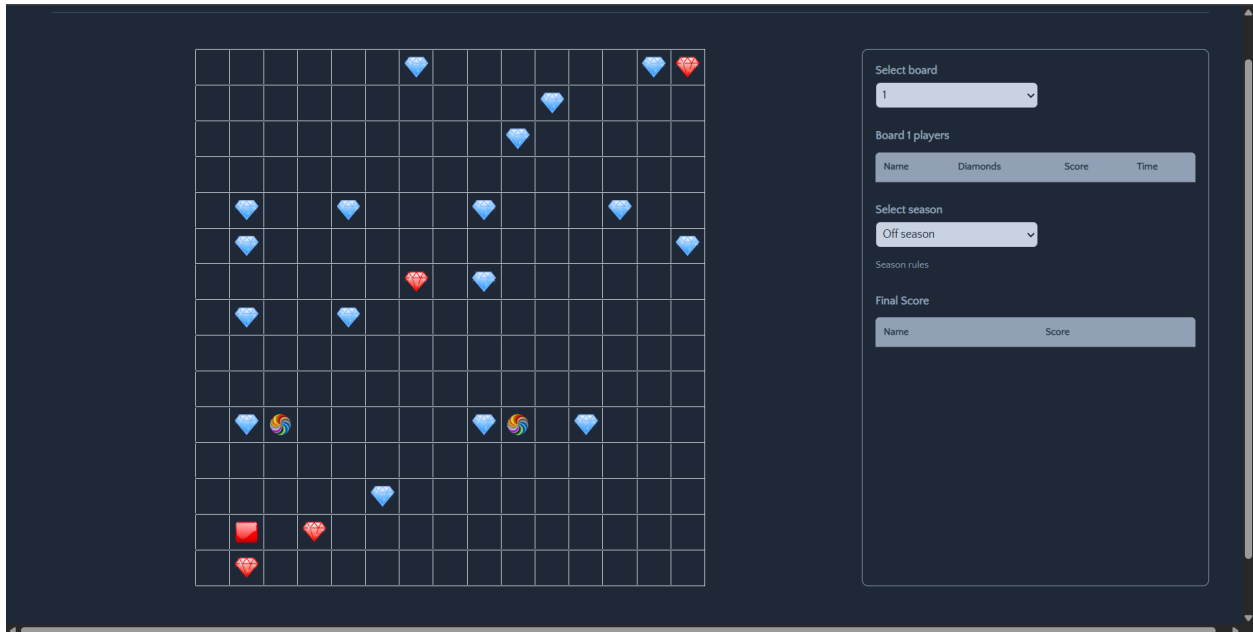


posisi bot saat ngespawn

Final Score	
Name	Score
ari	13

pada saat bot ngespawn, ia mengambil beberapa diamond terdekat terlebih dahulu lalu kembali ke base setelah itu mencari diamond lagi ke lokasi yang lebih jauh tetapi tetap mengambil diamond dalam radius terdekat dan mengutamakan diamond merah

2. Pengujian gacorbobot melawan random bot (bot dari pabrik starternya)



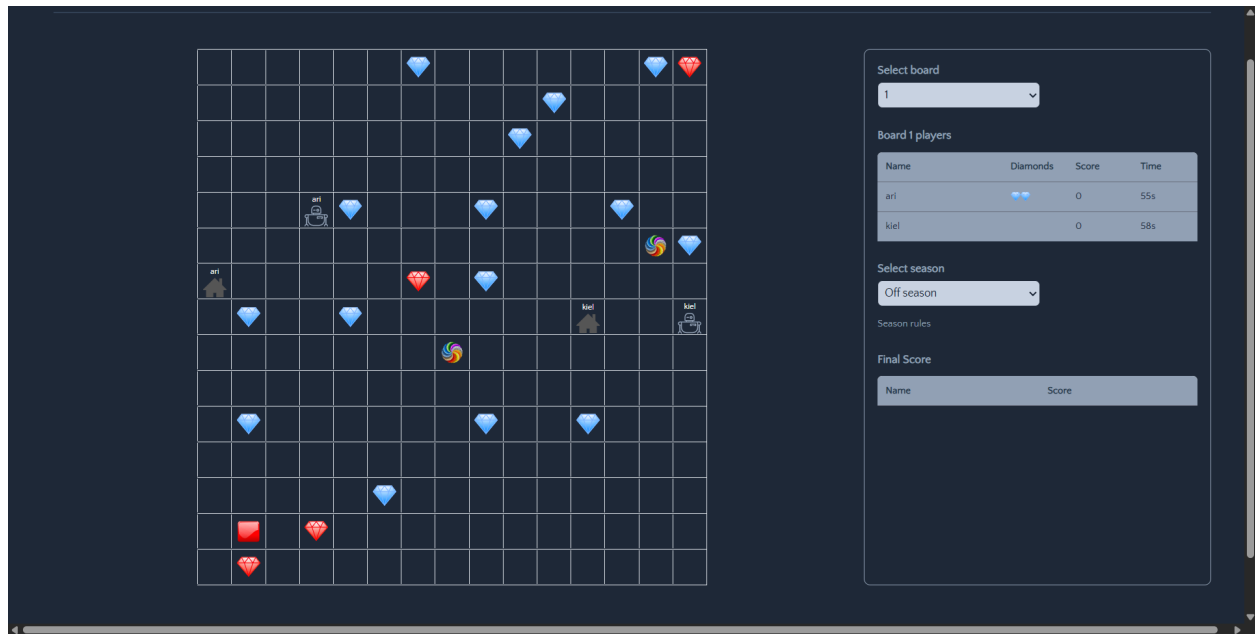
tampilan awal map sebelum ada bot spawn

```
C:\Users\Hezkiel\tubes1-IF2211-bot-starter-pack-1.0.1>python main.py --logic Random --email=your_email@kiel.com --name=kiel --password=awe --team rian
```

random bot cmd

```
C:\Users\Hezkiel\tubes1-IF2211-bot-starter-pack-1.0.1>python main.py --logic gacorbobot --email=kiel@gmail.com --name=ari --password=anjay --team etimo
```

gacorbobot cmd



posisi respawn bot

Final Score	
Name	Score
ari	8
kiel	0

final score dari kedua bot

gacorbob lebih baik dalam pengumpulan diamond dibandingkan dengan random bot (bot bawaan)

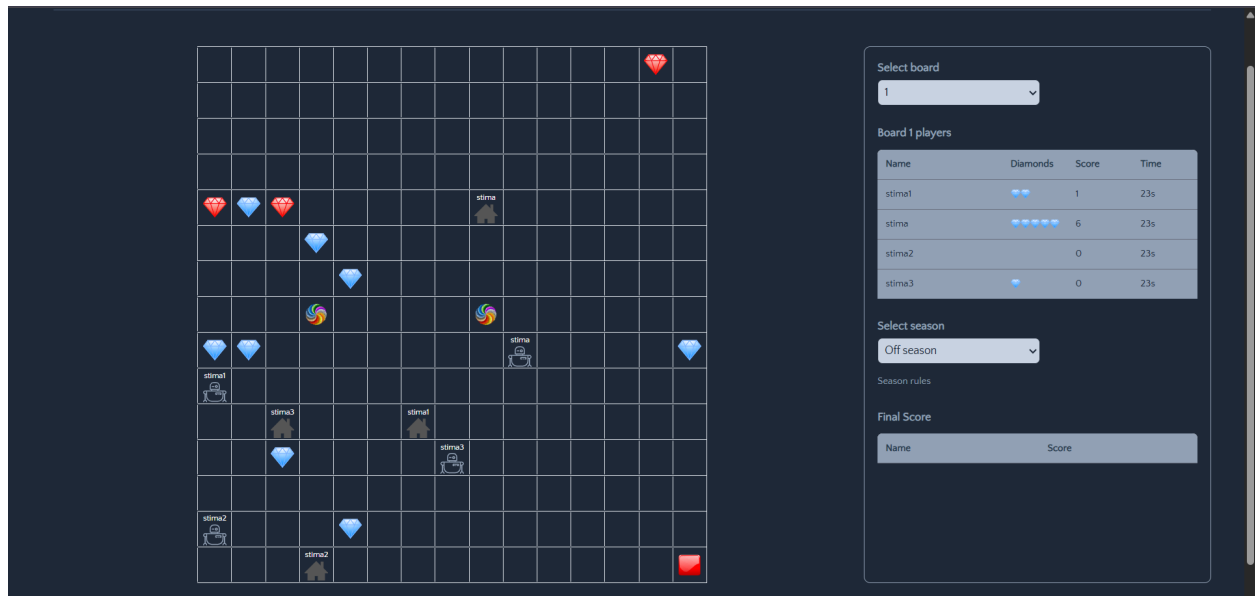
3. gacorbob vs 3 random bot



tampilan awal map sebelum bot spawn

```
run-bots.bat
1 @echo off
2 start cmd /c "python main.py --logic gacorbob --email=test@email.com --name=stima --password=123456 --team etimo"
3 start cmd /c "python main.py --logic Random --email=test1@email.com --name=stima1 --password=123456 --team etimo"
4 start cmd /c "python main.py --logic Random --email=test2@email.com --name=stima2 --password=123456 --team etimo"
5 start cmd /c "python main.py --logic Random --email=test3@email.com --name=stima3 --password=123456 --team etimo"
6
```

menggunakan command agar dapat mengaktifkan 4 bot sekaligus



posisi spawn bot. yang termasuk gacorbobot (stima) sisa nya random bot

Name	Score
stima	16
stima1	1
stima2	0
stima3	0

final score gacorbobot melawan 3 random bot dengan hasil gacorbobot sangat baik dan mendapati peringkat 1 dari ketiga bot, gacorbobot sangat cepat dalam mengumpulkan diamond terutama diamond merah dan tidak terlalu fokus untuk mengeliminasi musuh kecuali musuh didalam radius gacorbobot

4.3.2. Hasil Pengujian dan Analisis

Pengujian yang telah dilakukan menunjukkan bahwa gacorbot mampu beradaptasi dan menunjukkan kinerja yang baik dalam berbagai skenario permainan Diamonds. Berikut adalah hasil pengujian dan analisisnya:

1. Pengujian Jumlah Pengambilan Diamond (Gacorbot Tunggal)

Dalam pengujian ini, gacorbot diaktifkan sebagai bot tunggal di papan permainan. Hasilnya menunjukkan bahwa gacorbot memiliki strategi yang efektif dalam mengumpulkan diamond. Saat spawn, bot mengambil beberapa diamond terdekat terlebih dahulu sebelum kembali ke base untuk menyimpan perolehannya. Setelah itu, bot akan mencari diamond lagi ke lokasi yang lebih jauh, tetapi tetap memprioritaskan diamond dalam radius terdekat dan mengutamakan diamond merah. Ini memperlihatkan implementasi strategi greedy berbasis aturan yang mengutamakan keamanan (kembali ke base dengan diamond yang sudah terkumpul) dan efisiensi (memilih diamond terdekat dan memprioritaskan yang bernilai lebih tinggi).

2. Pengujian Gacorbot Melawan Random Bot (Bot Bawaan)

Skenario ini mempertandingkan gacorbot melawan random bot (bot bawaan dari starter pack). Hasil akhirnya menunjukkan bahwa gacorbot memiliki performa yang jauh lebih unggul dalam pengumpulan diamond dibandingkan dengan random bot. gacorbot berhasil mendapatkan skor 8, sementara random bot mendapatkan skor 0. Performa gacorbot yang lebih baik ini dapat dijelaskan karena random bot bergerak secara acak tanpa strategi, sedangkan gacorbot menggunakan logika greedy untuk mengambil keputusan optimal, seperti memprioritaskan diamond terdekat dan strategis.

3. Pengujian Gacorbot vs 3 Random Bot

Pengujian yang lebih kompleks dilakukan dengan mempertandingkan gacorbot melawan tiga random bot sekaligus. Untuk mengaktifkan keempat bot ini secara bersamaan, digunakan batch command (run-bots.bat). Hasil pengujian menunjukkan bahwa gacorbot (dengan nama "stima") mampu menduduki peringkat pertama dengan skor yang signifikan, mengalahkan ketiga random

bot lainnya. gacorbob menunjukkan kecepatan yang sangat baik dalam mengumpulkan diamond, terutama diamond merah. Dalam skenario ini, gacorbob tidak terlalu fokus untuk mengeliminasi musuh, kecuali musuh berada dalam radius dekat. Ini mencerminkan fokus gacorbob pada pengumpulan diamond sebagai prioritas utama dalam strategi greedy yang diimplementasikan.

Analisis Keseluruhan:

Dari hasil pengujian, dapat disimpulkan bahwa gacorbob telah berhasil mengimplementasikan strategi greedy berbasis aturan yang kompleks dan efektif. Logika pengambilan keputusannya secara efektif memetakan berbagai kondisi permainan "Diamonds" ke tindakan spesifik yang didasarkan pada serangkaian prioritas target. Strategi yang diterapkan mencakup manajemen inventory, prioritas pengumpulan jenis diamond yang berbeda (merah atau biru) dengan mempertimbangkan jarak dan jumlah diamond yang dibawa, serta fokus pada target di sekitar area base. Bot ini juga dirancang untuk memanfaatkan fitur peta seperti tombol merah dan teleporter untuk keuntungan.

Dari sisi efisiensi komputasi, gacorbob menunjukkan kinerja yang cepat dan responsif. Keunggulan ini dapat dicapai karena gacorbob menggunakan iterasi terbatas pada target atau objek permainan dengan kalkulasi jarak terdekat, memungkinkan keputusan cepat di setiap giliran. Namun, ada beberapa keterbatasan, seperti ketiadaan penghindaran rintangan proaktif, kurangnya mekanisme penghindaran lawan langsung, dan potensi terjebak dalam keputusan optimal lokal karena sifat algoritma greedy. Meskipun demikian, gacorbob menunjukkan perilaku yang rasional dalam berbagai skenario umum.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang dapat ditarik ialah program logika bot yang kami beri nama gacorbob telah berhasil mengimplementasikan strategi greedy berbasis aturan yang kompleks. Logika pengambilan keputusan secara efektif memetakan berbagai kondisi pada permainan “Diamonds” hingga pengambilan tindakan atau langkah spesifik yang didasarkan pada serangkaian prioritas target yang telah ditentukan. Strategi yang diterapkan mencakup berbagai aspek penting dalam permainan, yaitu, manajemen inventory dengan beberapa kondisi sebelum kembali ke base, prioritas dalam pengumpulan jenis diamond yang berbeda merah atau biru dengan mempertimbangkan jarak dan jumlah diamond yang dibawa, hingga memfokuskan pada target yang berada di area sekitar base terlebih dahulu. Program logika bot juga dirancang untuk bermain secara oportunistik dengan memanfaatkan fitur pada peta, seperti, tombol merah dan teleporter guna mendapatkan keunggulan.

Dari sisi efisiensi komputasi, gacorbob menunjukkan kinerja yang cenderung cepat dan responsif. Keunggulan ini dapat dicapai karena program logika bot menggunakan iterasi yang terbatas pada target atau objek permainan dengan kalkulasi jarak terdekat. Hal ini membuat bot dapat mengambil keputusan dengan cepat pada setiap giliran permainan.

Sementara untuk efektivitas, program logika bot dari gacorbob dapat berperilaku secara rasional dalam berbagai macam skenario yang umum. Akan tetapi, masih terdapat keterbatasan, yaitu, kapabilitas navigasi tanpa adanya penghindaran rintangan proaktif, ketiadaan mekanisme penghindaran lawan yang secara langsung mengatur di dalam alur logika utamanya. Sifat algoritma greedy secara umum juga dapat membuat program bot terjebak dalam keputusan optimal secara lokal akan tetapi tidak optimal secara global. Oleh karena itu, program logika bot masih bisa diperbaiki dan dikembangkan sehingga menghasilkan strategi algoritma yang efektif dan efisien untuk mencapai skor tertinggi dalam permainan.

5.2 Saran

Program logika bot bukanlah program algoritma greedy yang sempurna, sehingga membutuhkan saran-saran perbaikan, sebagai berikut:

1. Implementasi Pathfinding Sederhana.

Fungsi `peroleh_jarak` pada program logika bot dapat diganti dengan algoritma pathfinding yang lebih baik, misalnya, A* atau Breadth-First Search untuk navigasi. Algoritma pathfinding membuat bot dapat menghindari rintangan secara efektif dan menemukan jalur terpendek sebenarnya.

2. Mekanisme Penghindaran Lawan.

Program logika bot pada `gacorb` perlu ditambahkan logika untuk mendeteksi keberadaan bot lawan di sekitar bot milik kita sendiri atau di sekitar target seperti `diamond` merah atau biru sehingga bot dapat memilih jalur alternatif atau target alternatif apabila ada risiko untuk bertemu bot lawan.

3. Fleksibilitas Aturan dan Parameterisasi.

Daripada menggunakan nilai ambang batas yang kaku, sebaiknya program logika bot diprogram menggunakan parameter yang lebih dinamis.

4. Perbaikan Logika Teleporter.

Program logika bot bisa mengevaluasi semua kemungkinan pasangan teleporter untuk menemukan rute optimal, daripada hanya mengevaluasi teleporter terdekat dari bot atau base saja.

5. Integrasi Strategi Ofensif/Defensif.

Memperbaiki `kejar_bot_musuh` dengan lebih baik ke dalam alur keputusan utama sehingga bot dapat bergerak secara ofensif dan defensif sesuai dengan keadaan manajemen inventory atau jarak antara bot yang ada di papan.

LAMPIRAN

A. Repository Github ([link](#))

DAFTAR PUSTAKA

- [1] Khairunnisa, S.Pd., M.Cs, Nurhadi, S.Kom., M.Kom, A. R. Jatmiko, S.Si., M.Kom, Legito, ST., M.Kom, E. A. Saputra, S.T., M.T, F. Syafa'at, S.Kom., M.Kom, D. F. Surianto, S.Kom., M.Kom, R. Komalasari, S.Si., M.Kom, I. R. Mukhlis, S.Kom., M.Kom, Sulistyowati, S.T., M.Kom, T. A. Lorosae, M.Kom dan N. N. L. E. Zain, S.Si, BUKU AJAR LOGIKA & ALGORITMA, Jambi: PT. Sonpedia Publishing Indonesia, 2023.

- [2] F. Ikhwani dan H. Kurniawan, “Pemanfaatan Algoritma Greedy Pada Aplikasi Pemesanan Jasa Grooming Kucing Berbasis Android,” *IJCCS*, p. 3, 2019.

- [3] S. Oktaviana dan A. Naufal, “Algoritma Greedy untuk Optimalisasi Ruangan dalam Penyusunan Jadwal Perkuliahan,” *Jurnal Politeknik Negeri Jakarta*, p. 2, 2017.

