IMPLEMENTASI ALGORITMA GREEDY LOCAL OPTIMUM 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 4 (MOMOGI)

Maxavier Girvanus Manurung	123140191
Gilang Surya Agung	123140187
Muhammad Rafiq Ridho	123140197

Dosen Pengampu: Winda Yulita, M.Cs.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Diamonds merupakan suatu programming challenge yang mempertandingkan bot yang dibuat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi greedy dalam membuat bot ini.

Program permainan Diamonds terdiri atas:

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

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

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

Komponen-komponen dari permainan Diamonds antara lain:

1. **Diamonds**

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

2. **Red Button/Diamond Button**

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

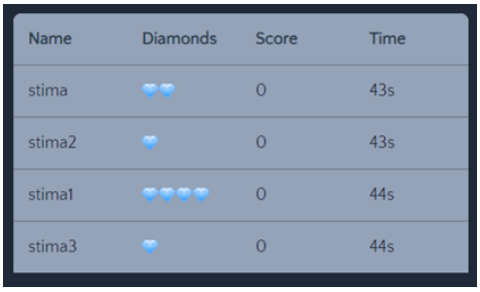
3. **Teleporters**

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

4. **Bots and Bases**

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

5. **Inventory**



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

Untuk mengetahui flow dari game ini, berikut ini adalah cara kerja permainan Diamonds.

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

Panduan Penggunaan

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

Mekanisme Teknis Permainan Diamonds

Permainan ini merupakan permainan berbasis web, sehingga setiap aksi yang dilakukan – mulai dari mendaftarkan bot hingga menjalankan aksi bot – akan memerlukan HTTP request terhadap

API endpoint tertentu yang disediakan oleh backend. Berikut adalah urutan requests yang terjadi dari awal mula permainan.

1. Program bot akan mengecek apakah bot sudah terdaftar atau belum, dengan mengirimkan POST request terhadap endpoint /api/bots/recover dengan body berisi email dan password bot. Jika bot sudah terdaftar, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut. Jika tidak, backend akan memberikan response code 404.
2. Jika bot belum terdaftar, maka program bot akan mengirimkan POST request terhadap endpoint /api/bots dengan body berisi email, name, password, dan team. Jika berhasil, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut.
3. Ketika id bot sudah diketahui, bot dapat bergabung ke board dengan mengirimkan POST request terhadap endpoint /api/bots/{id}/join dengan body berisi board id yang diinginkan (preferredBoardId). Apabila bot berhasil bergabung, maka backend akan memberikan response code 200 dengan body berisi informasi dari board.
4. Program bot akan mengkalkulasikan move selanjutnya secara berkala berdasarkan kondisi board yang diketahui, dan mengirimkan POST request terhadap endpoint /api/bots/{id}/move dengan body berisi direction yang akan ditempuh selanjutnya (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Apabila berhasil, maka backend akan memberikan response code 200 dengan body berisi kondisi board setelah move tersebut. Langkah ini dilakukan terus-menerus hingga waktu bot habis. Jika waktu bot habis, bot secara otomatis akan dikeluarkan dari board.
5. Program frontend secara periodik juga akan mengirimkan GET request terhadap endpoint /api/boards/{id} untuk mendapatkan kondisi board terbaru, sehingga tampilan board pada frontend akan selalu ter-update.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy adalah algoritma yang membuat pilihan terbaik secara lokal dengan harapan bahwa pilihan-pilihan lokal tersebut akan mengarah pada solusi global yang optimal. Algoritma ini selalu membuat pilihan yang terlihat terbaik. Meskipun algoritma ini memberikan solusi optimal di banyak masalah, algoritma ini tidak selalu memberikan solusi yang optimal[1].

2.2 Cara Kerja Program

Algoritma greedy yang digunakan oleh bot ini mengambil keputusan terbaik di setiap langkah berdasarkan kondisi saat itu, tanpa memperhitungkan dampak jangka panjang. Bot secara konsisten memilih diamond terdekat sebagai target utama dan bergerak langsung menuju posisinya untuk mengoptimalkan pengumpulan diamond secara cepat dan efisien. Jika inventory bot sudah penuh, bot segera kembali ke base untuk menyimpan diamond dan bersiap mengumpulkan diamond lagi. Dalam beberapa situasi, bot juga memanfaatkan fitur teleport jika terbukti dapat mempersingkat jarak menuju target seperti diamond atau base, demi meningkatkan efisiensi gerakan. Saat bot menentukan arah gerakan, bot memastikan bahwa langkah yang dipilih valid dan sesuai aturan permainan. Jika langkah pertama tidak memungkinkan, bot akan mencari alternatif gerakan lain. Pemilihan langkah dilakukan berdasarkan perhitungan jarak yang bertujuan untuk selalu mendekat ke target, menjadikan keputusan bersifat cepat dan lokal. Namun, bot tidak mempertimbangkan strategi jangka panjang seperti memilih diamond yang lebih bernilai tetapi lebih jauh, sehingga fokus utamanya adalah pada keuntungan langsung di setiap langkah.

1. Cara Implementasi Program

Program bot ini dirancang untuk bermain dalam sebuah game di mana tujuannya adalah mengumpulkan diamonds dan membawanya kembali ke base. Logika utama bot terdapat dalam metode next_move. Setiap kali giliran bot, metode ini pertama-tama akan menginisialisasi data teleporter jika belum ada. Kemudian, ia akan memeriksa apakah inventory bot sudah penuh, jika sudah, bot akan memprioritaskan kembali ke markas menggunakan fungsi move_to_base. Jika inventaris belum penuh, bot akan mencari berlian terdekat melalui find_target_diamond. Jika berlian ditemukan, bot akan bergerak menuju berlian tersebut menggunakan move_to_diamond. Apabila tidak ada berlian yang bisa dituju, bot akan kembali bergerak menuju base.

Fungsi find_target_diamond memiliki strategi khusus, yaitu ia akan mencari berlian standar (yang tidak memiliki pair_id). Namun, jika bot sudah memiliki 4 berlian, ia akan memprioritaskan pencarian berlian yang bernilai 1 poin (berlian biru).

Metode safe_move dan validate_move bertanggung jawab untuk memastikan pergerakan bot valid dan aman. safe_move menentukan arah pergerakan (atas, bawah, kiri, atau kanan) menuju target, dengan prioritas pada sumbu yang memiliki jarak lebih

besar. Jika pergerakan prioritas tidak valid, ia akan mencoba sumbu alternatif. validate_move kemudian memastikan bahwa langkah yang dipilih adalah salah satu dari empat arah mata angin yang valid dan diperbolehkan oleh papan permainan. Jika tidak ada langkah yang valid, bot akan tetap diam. Fungsi pendukung lainnya seperti distance, is_adjacent, dan direction_to membantu dalam kalkulasi jarak dan arah, tapi tidak berfungsi.

2. **Menjalankan Bot Program**

Untuk menjalan bot pertama-tama pada main.py lakukan import file logic terlebih dahulu disini file logic saya bernama selfbot, kemudian masukkan kelas logic yang telah di import kedalam controller dan berikan nama pada kelas logicnya, disini nama kelas logicsaya Greedy.

```
import argparse
from time import sleep

from colorama import Back, Fore, Style, init
from game.api import Api
from game.board_handler import BoardHandler
from game.bot_handler import BotHandler
from game.logic.random import RandomLogic
from game.util import *
from game.logic.base import BaseLogic
from game.logic.selfbot import MyBot

init()
BASE_URL = "http://localhost:3000/api"
DEFAULT_BOARD_ID = 1
CONTROLLERS = {
    "Random": RandomLogic, "Greedy": MyBot
}
```

Setelah itu buka CMD dan masuk ke directory game engine dan lakukan npm run start, lalu masuk/klik pada lokalhost berikut http://localhost:8082/. Selanjut nya buka CMD satu lagi dan masuk ke directory bot starter pack dan untuk menjalankan satu bot dapat menggunakan python main.py --logic Greedy --email=your_email@example.com --name=your_name --password=your_password --team etimo, ganti bagian berwarna kuning sesuai key pada controller. Untuk menjalankan banyak bot buka run-bots.bat untuk windows dan run-bots.sh untuk linux, ganti bagian bagian logic dengan key pada controller dan run.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Proses ini dimulai dengan inisialisasi daftar teleporters melalui metode init_teleporters, di mana bot mencari semua objek yang bertipe TeleportGameObject dan menyimpannya dalam atribut self.teleporters. Selanjutnya, dalam metode find_target_diamond, bot melakukan mapping terhadap diamond yang tersedia di papan permainan dengan memfilter diamond yang tidak memiliki pasangan. Proses mapping berlanjut di metode next_move, di mana bot menentukan langkah selanjutnya berdasarkan informasi yang telah dikumpulkan. Jika inventori bot sudah penuh, bot akan memetakan jalur kembali ke base. Sebaliknya, jika inventori belum penuh, bot akan bergerak menuju diamond terdekat yang telah diidentifikasi sebelumnya. Sebelum melakukan gerakan, bot memetakan kemungkinan gerakan yang valid melalui metode safe_move dan validate_move. Metode ini memastikan bahwa semua langkah yang diambil oleh bot adalah aman dan sesuai dengan aturan permainan, dengan memeriksa apakah gerakan yang diusulkan berada dalam batas papan permainan dan diperbolehkan. Jika gerakan tidak valid, bot akan mencari alternatif gerakan yang masih memungkinkan.

3.2 Eksplorasi Alternatif Solusi Greedy

Alternatif solusi greedy yang kami pikirkan adalah Local optimum karena pada solusi ini akan mempertimbangkan respon cepat terhadap nilai yang lebih tinggi maupun lebih rendah, baik dari segi jarak, maupun nilai dari value diamonds.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

1. Local Optimum

   Local optimum adalah solusi yang pada setiap langkahnya akan memilih solusi yang paling optimal, baik secara maksimum maupun secara minimum tanpa mempertimbangkan langkah berikutnya[2].

   a. Local maximum

      Local maximum adalah jenis spesifik dari local optimum dimana akan terjadi ketika tujuan utamanya adalah memaksimal sesuatu, sehingga setiap langkahnya akan memilih solusi yang paling besar.

   b. Local minimum

      Local maximum adalah jenis spesifik dari local optimum dimana akan terjadi ketika tujuan utamanya adalah meminimalkan sesuatu, sehingga setiap langkahnya akan memilih solusi yang paling rendah.

3.4 Strategi Greedy yang Dipilih

Program ini menggunakan pendekatan algoritma greedy yang berfokus pada pengambilan keputusan lokal optimal(local optimum) pada setiap langkah untuk memaksimalkan efisiensi jangka pendek. Strategi utamanya adalah selalu menargetkan diamond terdekat berdasarkan perhitungan jarak Euclidean, dengan pertimbangan khusus saat kapasitas inventory >= 4 dimana bot akan selektif hanya mengambil diamond biru (1 point) untuk menghindari error jika di dekatnya terdapat diamond merah (2 point).

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
# Maxavier Girvanus Manurung (123140191)
# Gilang Surya Agung (123140187)
# Muhammad Rafiq Ridho (123140197)
# Kelompok: 4 Momogi

from typing import Tuple, List
import math
from game.models import Board, GameObject, Position
from game.logic.base import BaseLogic

class MyBot(BaseLogic):
    def __init__(self):
        self.teleporters: List[GameObject] = []  # menyimpan data
teleport
        self.teleport_pairs = {}  # menyimpan data pasangan teleport
    # fungsi untuk menentukan langkah selanjutnya
    def next_move(self, bot: GameObject, board: Board) -> Tuple[int,
int]:
        current_pos = bot.position # mengambil posisi bot saat ini
        props = bot.properties # mengambil proprti dari bot

        if not self.teleporters:
            self.init_teleporters(board)

        if props.diamonds == props.inventory_size: # kembali ke base
jika inventory penuh
            return self.move_to_base(current_pos, props.base, board)

        target_diamond = self.find_target_diamond(bot, board) #
menemukan diamond terdekat
        if target_diamond:
            return self.move_to_diamond(current_pos,
target_diamond.position, board)
        else:
            return self.move_to_base(current_pos, props.base, board)
# Jika tidak ada diamond yang ditemukan, bergerak ke arah base

    def init_teleporters(self, board: Board) -> None: #
        self.teleporters = [obj for obj in board.game_objects if
obj.type == "TeleportGameObject"]
```

```
        if len(self.teleporters) >= 2:
            self.teleport_pairs = {
                self.teleporters[0].id: self.teleporters[1],
                self.teleporters[1].id: self.teleporters[0]
            }
    # fungsi untuk kembali ke base
    def move_to_base(self, current_pos: Position, base_pos:
Position, board: Board) -> Tuple[int, int]:
        for tele in self.teleporters: # mencari teleport yang ada
            if tele.id in self.teleport_pairs:
                tele_dest = self.teleport_pairs[tele.id]  #
                if self.distance(tele_dest.position, base_pos) <
self.distance(current_pos, base_pos):
                    if self.is_adjacent(current_pos, tele.position):
                        return self.direction_to(current_pos,
tele.position)
        return self.safe_move(current_pos, base_pos, board)

    # fungsi untuk bergerak ke diamond terdekat
    def move_to_diamond(self, current_pos: Position, diamond_pos:
Position, board: Board) -> Tuple[int, int]:
        for tele in self.teleporters: # mencari pasangan teleport
untuk bergerak ke diamond
            if tele.id in self.teleport_pairs:
                tele_dest = self.teleport_pairs[tele.id]
                if self.distance(tele_dest.position, diamond_pos) <
self.distance(current_pos, diamond_pos): # menghitung jarak
menggunakan tele dan tidak
                    if self.is_adjacent(current_pos, tele.position):
                        return self.direction_to(current_pos,
tele.position) # bergerak ke teleport
        return self.safe_move(current_pos, diamond_pos, board) #
bergerak ke diamond terdekat tanpa tele jika tidak ada tele terdekat

    def find_target_diamond(self, bot: GameObject, board: Board) ->
GameObject: # fungsi untuk mencari diamond
        diamonds = [d for d in board.diamonds if not
d.properties.pair_id]

        if bot.properties.diamonds >= 4: # jika inventory berisi 4
maka bot mengambil diamond biru saja
            diamonds = [d for d in diamonds if d.properties.points
== 1]

        return min(diamonds, key=lambda d:
self.distance(bot.position, d.position)) if diamonds else None
```

```
    # fungsi bot untuk bergerak kanan kiri atas atau bawah
    def safe_move(self, current_pos: Position, target_pos: Position,
board: Board) -> Tuple[int, int]:
        dx = 1 if target_pos.x > current_pos.x else -1 if
target_pos.x < current_pos.x else 0 # bergerak ke atas atau bawah
        dy = 1 if target_pos.y > current_pos.y else -1 if
target_pos.y < current_pos.y else 0 # bergerak ke kiri atau kanan
        # melakukan validasi gerakan valid atau tidak
        if abs(target_pos.x - current_pos.x) > abs(target_pos.y -
current_pos.y):
            dy = 0
        else:
            dx = 0

        if not board.is_valid_move(current_pos, dx, dy):
            if dx != 0:
                dy = 1 if target_pos.y > current_pos.y else -1 if
target_pos.y < current_pos.y else 0
                dx = 0
            else:
                dx = 1 if target_pos.x > current_pos.x else -1 if
target_pos.x < current_pos.x else 0
                dy = 0

        return self.validate_move(current_pos, dx, dy, board)

    def validate_move(self, pos: Position, dx: int, dy: int, board:
Board) -> Tuple[int, int]:
        valid_moves = {(1,0), (0,1), (-1,0), (0,-1)}
        if (dx, dy) in valid_moves and board.is_valid_move(pos, dx,
dy): # melakukan validasi gerakan apakah di perbolehkan atau tidak
            return (dx, dy)

        for move in valid_moves: # melakukan iterasi semua gerakan
dan mengambil gerakan yang valid
            if board.is_valid_move(pos, move[0], move[1]):
                return move
        return (0, 0) # tidak bergerak jika tidak valid
    # menghitung jarak teleport
    def distance(self, pos1: Position, pos2: Position) -> float:
        return math.sqrt((pos1.x - pos2.x)**2 + (pos1.y -
pos2.y)**2)

    def is_adjacent(self, pos1: Position, pos2: Position) -> bool:
        return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y) == 1
    # menghitung jarak dari posisi sekarang ke posisi target
```

```
    def direction_to(self, current_pos: Position, target_pos:
Position) -> Tuple[int, int]:
        dx = target_pos.x - current_pos.x
        dy = target_pos.y - current_pos.y
        return (dx, dy) if self.is_adjacent(current_pos, target_pos)
else (0, 0)
```

2. **Penjelasan Alur Program**

```
class MyBot(BaseLogic):
```

Pertama melakukan inisialisasi class MyBot yang mewarisi BaseLogic dan kemudian dibuat 2 kontructor yang menyimpan 2 variabel data teleport.

```
def next_move(self, bot: GameObject, board: Board) -> Tuple[int,
int]:
```

Kemudian pada fungsi next_move bot akan mengambil posisi dan properti bot seperti inventory saat ini. Kemudian melakukan cek apakah inventori penuh atau tidak jika tidak bot akan mencari target diamond. Dan jika terdapat 4 diamond bot akan mencari diamond yang bernilai 1 poin. Jika target diamond di temukan bot akan bergerak mengarah ke diamond tersebut dengan memanggil move_to_diamond

```
def move_to_base(self, current_pos: Position, base_pos: Position,
board: Board) -> Tuple[int, int]:
```

Pertama tama bot akan mnegecek apakah terdapat teleport yang dapat digunakan untuk kembali ke base yang berdekatan dengan bot. Jika ada bot akan bergerak ke mengarah ke teleport. Jika tidak ada bot akan bergerak dengan memanggil save_move.

```
def move_to_diamond(self, current_pos: Position, diamond_pos:
Position, board: Board) -> Tuple[int, int]:
```

Disini untuk mendekati diamond bot akan mencari teleport apakah terdapat teleport yang dapat digunakan jika tidak ada bot akan bergerak mendekati diamond menggunakan safe_move.

```
 def find_target_diamond(self, bot: GameObject, board: Board) ->
GameObject: # fungsi untuk mencari diamond
```

Disini bot akan mengecek diamond yang terdapat dalam board untuk menemukan target diamond kemudian dicek ke inventory apakah diamond yang dimiliki >= 4 jika iya bot akan mencari diamond bernilai 1 point terdekat berdasarkan jarak dan posisi saat ini.

```
def safe_move(self, current_pos: Position, target_pos: Position,
board: Board) -> Tuple[int, int]:
```

Untuk save_move bot akan bergerak dengan menentukan arah dx, dy berdasarkan posisi target dengan memastikan gerakan yang diusulkan valid dengan memeriksa apakah gerakan tersebut berada dalam batas board. Jika gerakan tidak valid, bot akan mencari alternatif gerakan yang masih memungkinkan.

```
def validate_move(self, pos: Position, dx: int, dy: int, board:
Board) -> Tuple[int, int]:
```

Pada validasi move diperiksa apakah gerakan yang diusulkan valid atau tidak jika iya akan mengembalikan gerakan tersebut, jika tidak valid akan mencoba semua kemungkinan dan mengembalikan gerakan yang valid pertama yang ditemukan.

```
        def direction_to(self, current_pos: Position, target_pos: Position) -> Tuple[int, int]:
```

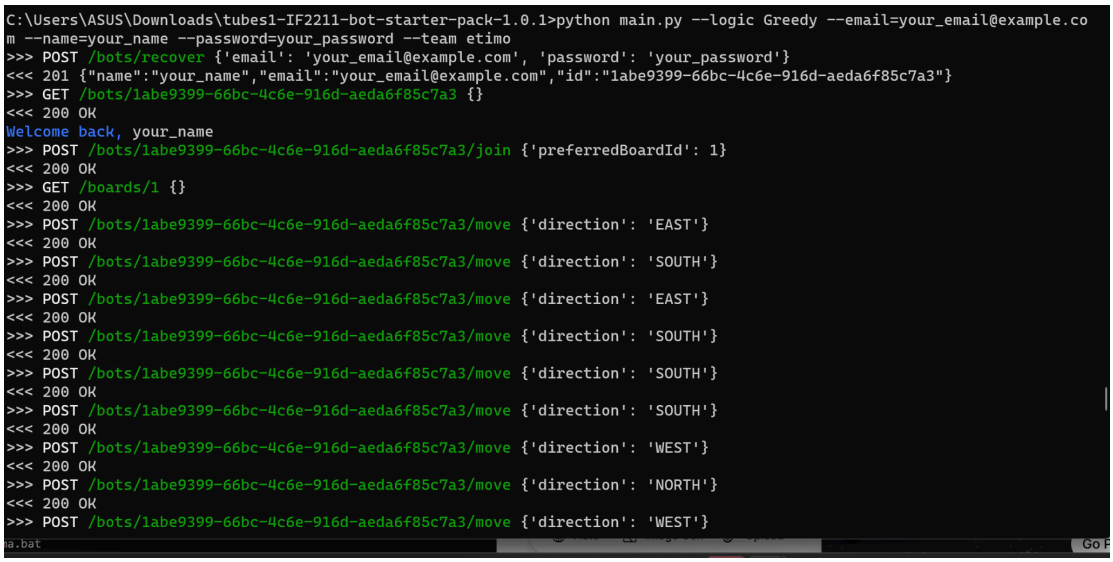
Menghitung arah dari posisi saat ini ke posisi target dan mengembalikan perubahan posisi (dx, dy) jika posisi target berdekatan.

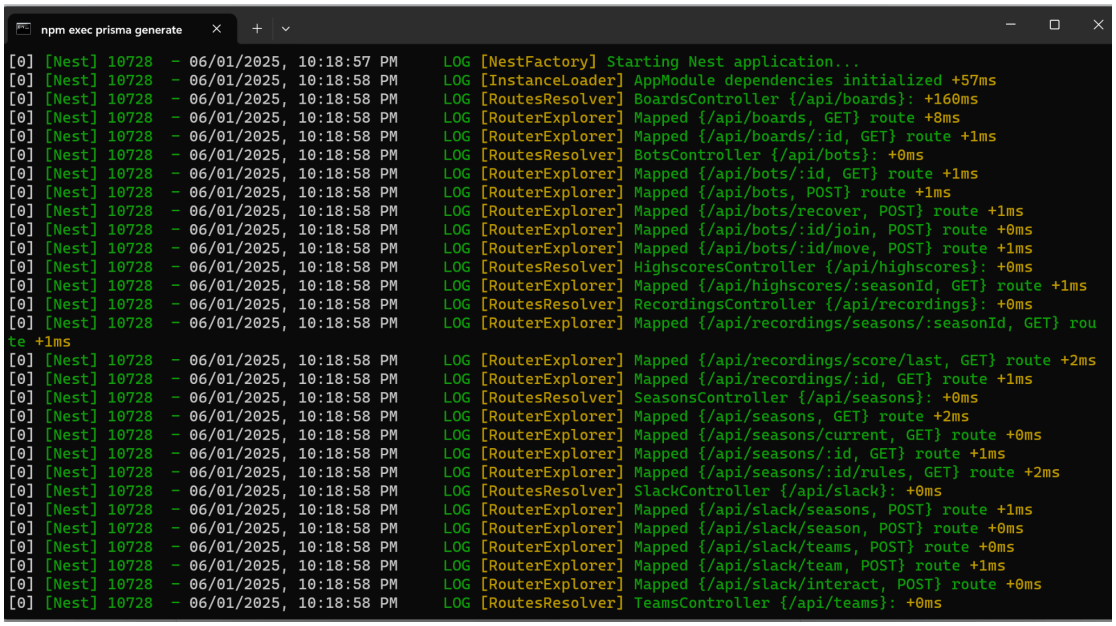
4.2 Struktur Data yang Digunakan

a. List pada self.teleporters (menyimpan data teleport) dan diamond (menyimpan data diamond yang tersedia dalam board)
b. Dictionary pada self.teleport_pair menyimpan pasangan teleport
c. Tuple pada next_move, move_to_base, move_to_diamond, safe_move.

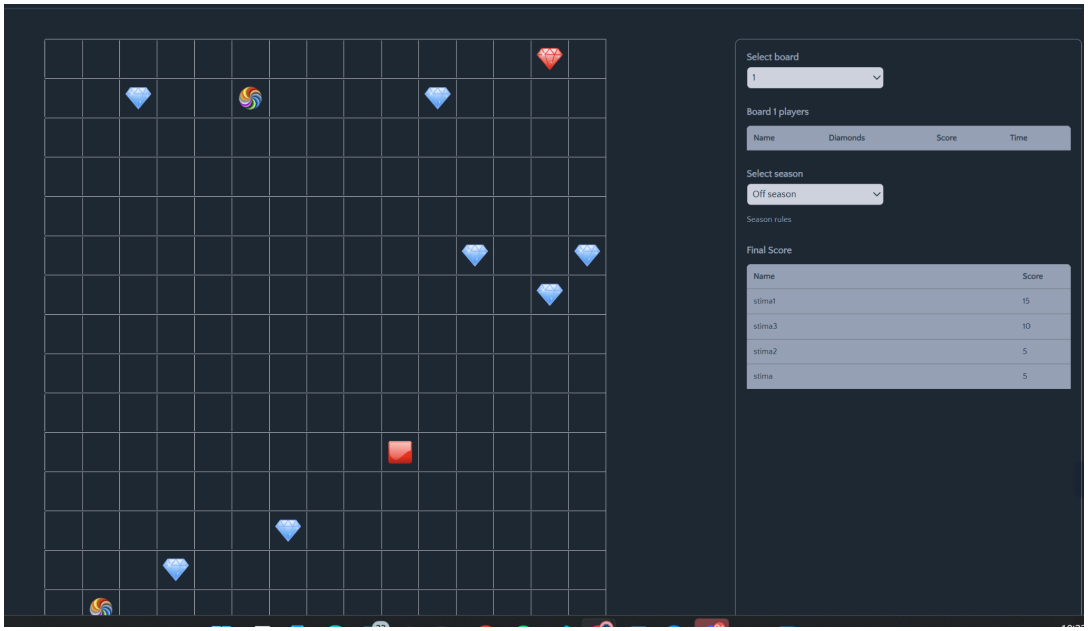
4.3 Pengujian Program

1. Skenario Pengujian





2. Hasil Pengujian dan Analisis



Dari hasil tersebut ditunjukan bahwa logic bot menggunakan algoritma greedy untuk menentukan langkah yang memungkinkan pengambilan keputusan yang cepat untuk kondisi saat ini. Namun masih terdapat kekurangan seperti pemaanfaatan teleport yang kurang maksimal dan bot menghiraukan obstakel yang ada di depannya untuk mencapai tujuan, sehingga jika terdapat diamond, bot dan pasangan teleport dalam garis lurus bot tersebut akan tetap bergerak lurus sehingga seperti terjadi loop.

BAB V
KESIMPULAN

5.1 Kesimpulan

Berdasarkan dari implementasi strategi greedy dalam permainan Diamond, dapat disimpulkan bahwa pendekatan ini mampu memberikan hasil yang efisien dalam pengambilan keputusan secara cepat. Strategi greedy bekerja dengan cara memilih solusi terbaik pada saat itu, seperti memilih diamond yang paling dekat dan kembali ke base saat inventory telah penuh. Pendekatan ini sangat cocok untuk game berbasis waktu karena dapat menghasilkan respons yang cepat dalam berbagai situasi.

Bot juga berhasil memanfaatkan elemen permainan tambahan seperti teleport untuk memperpendek jarak ke target, serta mengevaluasi setiap langkah agar tetap sesuai dengan peraturan permainan. Meskipun strategi ini tidak mempertimbangkan situasi jangka panjang, hasil yang diperoleh menunjukkan bahwa pendekatan greedy cukup efektif dalam konteks permainan ini.

LAMPIRAN

A. Repository Github

DAFTAR PUSTAKA

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
[2] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Harlow, England: Pearson Education Limited, 2021.