

**LAPORAN TUGAS BESAR I  
IF 2211 STRATEGI ALGORITMA**

**PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT  
PERMAINAN DIAMONDS**



**KELOMPOK SARIKATJAVA  
ANGGOTA:  
IMANUEL SEBASTIAN GIRSANG (13522058)  
VENANTIUS SEAN ARDI NUGROHO (13522078)  
JULIAN CHANDRA SUTADI (1322080)**

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>BAB I</b>	
<b>DESKRIPSI MASALAH.....</b>	<b>2</b>
<b>BAB II</b>	
<b>LANDASAN TEORI.....</b>	<b>4</b>
2.1. Algoritma Greedy.....	4
2.2. Elemen Algoritma Greedy.....	4
2.3. Struktur Program Keseluruhan.....	5
2.4. Game Engine Diamonds.....	6
2.5. Alur Pengembangan Bot.....	7
2.6. Alur Menjalankan Program.....	9
<b>BAB III</b>	
<b>APLIKASI STRATEGY GREEDY.....</b>	<b>12</b>
3.1. Alternatif Greedy.....	12
3.1.1. Greedy By Distance.....	12
3.1.2. Greedy By Points.....	13
3.1.3. Greedy By Point Density.....	14
3.1.4. Greedy By Area Density.....	16
3.2. Strategi Greedy yang Diimplementasikan.....	18
<b>BAB IV</b>	
<b>IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>19</b>
4.1. Implementasi dalam Pseudocode.....	19
4.2. Struktur Data Program.....	25
4.3. Analisis dan Pengujian.....	26
<b>BAB V</b>	
<b>SIMPULAN DAN SARAN.....</b>	<b>31</b>
5.1. Simpulan.....	31
5.2. Saran.....	31
<b>LAMPIRAN.....</b>	<b>33</b>
Repositori Github.....	33
Link Video Tugas Besar I Strategi Algoritma.....	33
Screenshot Hasil Testing.....	33
<b>DAFTAR PUSTAKA.....</b>	<b>37</b>

## BAB I

### DESKRIPSI MASALAH

Diamonds adalah suatu *programming challenge* yang akan mempertandingkan bot yang dibuat oleh seorang pemain dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya.

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.

Flow dari permainan Diamonds adalah sebagai berikut:

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

## BAB II

# LANDASAN TEORI

### 2.1. Algoritma Greedy

Algoritma Greedy adalah pendekatan heuristik dalam pemecahan masalah yang berfokus pada pengambilan keputusan secara lokal untuk setiap langkah, dengan harapan bahwa setiap langkah yang diambil akan mengarah pada solusi yang optimal secara keseluruhan. Prinsip dasar dari algoritma ini adalah memilih opsi terbaik yang tersedia pada setiap langkah, tanpa mempertimbangkan konsekuensi jangka panjang dari pilihan tersebut.

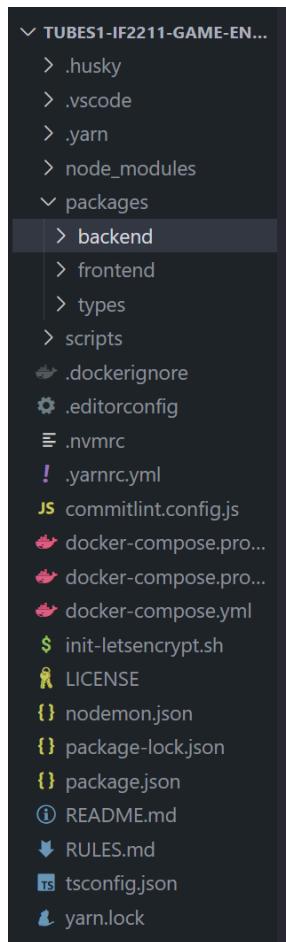
Implementasi algoritma Greedy seringkali cocok untuk masalah optimisasi yang dapat dipecahkan secara bertahap, di mana setiap langkah hanya bergantung pada langkah sebelumnya dan tidak ada ketergantungan silang antara langkah-langkah. Namun, penting untuk diingat bahwa tidak semua masalah cocok untuk pendekatan Greedy, karena keputusan yang diambil secara lokal mungkin tidak selalu mengarah pada solusi global yang optimal.

### 2.2. Elemen Algoritma Greedy

Terdapat enam elemen algoritma Greedy, antara lain:

1. Himpunan kandidat (C): berisi kandidat pilihan pada setiap langkah.
2. Himpunan solusi (S): yang berisi kandidat yang sudah dipilih
3. Fungsi solusi: menentukan apakah himpunan solusi sudah memberikan solusi
4. Fungsi seleksi: strategi heuristik untuk memilih kandidat berdasarkan strategi greedy tertentu.
5. Fungsi kelayakan: fungsi yang memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif: fungsi yang ingin dicapai nilai optimalnya.

### 2.3. Struktur Program Keseluruhan



Gambar 1. Struktur Program Game Engine Diamonds

Game engine bertindak sebagai penghubung antara pemain (bot) dan server backend dalam suatu permainan berbasis web. Terdapat beberapa folder utama yang menjadi kunci utama bahwa program bisa berjalan dengan baik.

1. Node\_modules : Folder ini berisi berbagai pustaka maupun modul dari NPM (*Node Package Manager*). Pustaka dan modul ini menjadi hal yang krusial agar keseluruhan kode program dari game engine dapat bekerja.
2. Packages: Folder ini berisi *source code* program dari sisi *backend* maupun *frontend*.
3. Types: Folder ini berisi kumpulan tipe data yang digunakan dalam kode program. Hal ini wajib dibuat dalam pemrograman dengan bahasa *typescript*.
4. Scripts: Folder ini berisi berbagai *script* yang harus dijalankan agar *game engine* dapat berjalan dengan baik.

Selain itu, terdapat beberapa file konfigurasi lain untuk membuat program dapat berjalan dengan baik seperti konfigurasi docker, package.json, serta nodemon.

*Entrypoint* untuk menjalakan bot terdapat pada file **main.py**. Pada file ini terdapat berbagai aturan untuk melakukan request terhadap *endpoint-endpoint* yang ada di *backend*. Didalam folder *game* terdapat berbagai file yang berfungsi sebagai logic untuk menyambungkan bot dengan *board*. Pada folder ini terdapat berbagai macam *handler* yang berfungsi untuk melakukan semacam setup agar bot kita bisa memperoleh informasi yang dibutuhkan untuk kemudian membuat keputusan kemana arah selanjutnya yang perlu diambil. Terakhir ada foler logic di dalam folder game yang dimana disinilah logic dari bot-bot akan diimplementasikan sebagai suatu class. Class ini nantinya akan dipanggil melalui **main.py** untuk disambungkan dengan API endpoint dari game engine.

## 2.4. Game Engine Diamonds

Proses dimulai dengan mendaftarkan bot, di mana bot mengirimkan permintaan POST ke endpoint */api/bots/recover* dengan menyertakan email dan password bot. Server backend kemudian memeriksa apakah bot sudah terdaftar atau belum. Jika bot sudah terdaftar sebelumnya, server akan memberikan respons dengan kode 200 bersamaan dengan ID bot yang terkait. Namun, jika bot belum terdaftar, respons yang diberikan oleh server akan berupa kode 404, yang menandakan bahwa bot perlu didaftarkan.

Jika bot belum terdaftar, langkah selanjutnya adalah bot melakukan permintaan POST ke endpoint */api/bots*, kali ini dengan memberikan informasi seperti email, nama, password, dan tim bot tersebut. Jika proses ini berhasil, server backend akan memberikan respons dengan kode 200 bersamaan dengan ID bot yang baru saja didaftarkan.

Setelah bot memiliki ID yang valid, langkah selanjutnya adalah bergabung ke dalam board permainan. Untuk melakukan hal ini, bot mengirimkan permintaan POST ke endpoint */api/bots/{id}/join*, dengan menyertakan ID bot dan ID board yang diinginkan (*preferredBoardId*). Server backend kemudian akan memproses permintaan tersebut. Jika bot berhasil bergabung, server akan memberikan respons dengan kode 200 serta informasi mengenai board yang telah bergabung.

Selanjutnya, bot akan secara berkala melakukan kalkulasi untuk menentukan langkah selanjutnya berdasarkan kondisi terkini dari board. Setelah melakukan kalkulasi, bot mengirimkan permintaan POST ke endpoint */api/bots/{id}/move*, dengan memberikan informasi

mengenai ID bot dan arah langkah yang akan diambil (misalnya: "NORTH", "SOUTH", "EAST", atau "WEST"). Server backend akan memproses permintaan tersebut dan memberikan respons dengan kode 200 serta informasi mengenai kondisi board setelah langkah tersebut diambil.

Di sisi frontend, terdapat juga proses pembaruan tampilan board secara berkala. Frontend akan melakukan permintaan GET ke endpoint `/api/boards/{id}` dengan tujuan untuk mendapatkan kondisi terbaru dari board. Dengan demikian, tampilan board pada frontend akan selalu ter-update sesuai dengan kondisi terbaru dari permainan.

Adapun waktu permainan bagi bot terbatas. Jika waktu bot habis, bot akan secara otomatis dikeluarkan dari board, sehingga proses permainan akan berlanjut tanpa kehadiran bot tersebut. Dengan demikian, melalui serangkaian langkah-langkah ini, game engine memungkinkan bot untuk berinteraksi dengan board game secara online melalui API yang disediakan oleh server backend, sementara frontend dapat memperbarui tampilan board secara periodik untuk memberikan pengalaman bermain yang dinamis bagi pengguna.

## 2.5. Alur Pengembangan Bot

Bot dikembangkan dengan cara membuat file-file baru pada folder logic. Pada setiap file, kita mendefinisikan sebuah kelas yang merupakan *inheritance* dari Class BaseLogic. Dari situ, kita dapat membuat method-method yang nantinya dapat membantu bot menentukan ke arah mana dia harus bergerak. Terdapat satu method yang wajib dimiliki oleh semua bot yaitu `next_move`. Method ini memberikan return ke arah mana bot harus bergerak (1,0 | -1,0 | 0,1 | 0,-1) yang merepresentasikan keempat arah mata angin.



```
1 import random
2 from typing import Optional
3
4 from game.logic.base import BaseLogic
5 from game.models import GameObject, Board, Position
6 from ..util import get_direction
7
8
9 class RandomLogic(BaseLogic):
10     def __init__(self):
11         self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
12         self.goal_position: Optional[Position] = None
13         self.current_direction = 0
14
15     def next_move(self, board_bot: GameObject, board: Board):
16         props = board_bot.properties
17         # Analyze new state
18         if props.diamonds == 5:
19             # Move to base
20             base = board_bot.properties.base
21             self.goal_position = base
22         else:
23             # Just roam around
24             self.goal_position = None
25
26         current_position = board_bot.position
27         if self.goal_position:
28             # We are aiming for a specific position, calculate delta
29             delta_x, delta_y = get_direction(
30                 current_position.x,
31                 current_position.y,
32                 self.goal_position.x,
33                 self.goal_position.y,
34             )
35         else:
36             # Roam around
37             delta = self.directions[self.current_direction]
38             delta_x = delta[0]
39             delta_y = delta[1]
40             if random.random() > 0.6:
41                 self.current_direction = (self.current_direction + 1) % len(
42                     self.directions
43                 )
44
45         return delta_x, delta_y
```

Gambar 2. Contoh kode program untuk membuat bot

Setelah bot selesai diimplementasikan, bot perlu untuk dimasukkan ke **main.py** untuk dapat digunakan dalam game.

```

1 import argparse
2 from time import sleep
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.util import *
10 from game.logic.base import BaseLogic
11 from game.logic.ShortestDistance import ShortestDistance
12 from game.logic.teleportGreed import teleportGreed
13 from game.logic.points import points
14 from game.logic.DensityBot import HeuristicDensityBot
15 from game.logic.PureDensity import PureDensityBot
16 from game.logic.SquareDensity import SquareDensity
17 from game.logic.SarikatJava import sarikatJava
18
19 init()
20 BASE_URL = "http://localhost:3000/api"
21 DEFAULT_BOARD_ID = 1
22 CONTROLLERS = {
23     "Random": RandomLogic,
24     "Short": ShortestDistance,
25     "Teleport" : teleportGreed,
26     "Point" : points,
27     "DensityBest": HeuristicDensityBot,
28     "DensityNaive": PureDensityBot,
29     "Square" : SquareDensity,
30     "SJ" : sarikatJava
31 }

```

Gambar 3. Contoh kode untuk pemanggilan bot

Dari sini, bot sudah dapat dimasukkan sebagai *arguments* dalam *command line* untuk kemudian dimainkan di website terkait.

## 2.6. Alur Menjalankan Program

Terdapat dua tahapan yang perlu untuk menjalankan permainan Etimo: Diamonds, yaitu menyalakan *game engine* dan menjalankan *bot*. Untuk menjalankan *game engine* perlu dilakukan hal-hal berikut :

- Download file zip yang ada pada [link](https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0)
- Download Node serta Docker (jika belum memiliki)
- Lakukan Instalasi sesuai dengan petunjuk yang ada di Repository terkait
- Setelah selesai, buka terminal
- Untuk menjalankan *engine*, masukkan perintah “*npm run start*” pada directory src

```

● PS C:\Users\ASUS ROG\Documents\Tubes1_Stima> cd tubes1-IF2211-game-engine-1.1.0
○ PS C:\Users\ASUS ROG\Documents\Tubes1_Stima\tubes1-IF2211-game-engine-1.1.0> npm run start

> start
> npm run support && concurrently "npm run start:backend" "npm run start:frontend"

> support
> docker-compose up -d database

[+] Running 1/0
✓ Container tubes1-if2211-game-engine-110-database-1  Running
[0]
[0] > start:backend
[0] > cd packages/backend && npm run start:dev
[0]
[1]
[1] > start:frontend
[1] > cd packages/frontend && npm run dev
[1]
[1]

```

Gambar 4. Perintah untuk menjalankan *game engine*

- Berikut adalah tampilan jika berhasil dijalankan

```

[0] [nodemon] 1.19.4
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching dir(s): dist\**\*.env
[0] [nodemon] watching extensions: ts, map, js, json
[0] [nodemon] starting nest start
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [NestFactory] Starting Nest application...
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [InstanceLoader] AppModule dependencies initialized +26ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] BoardsController {/api/boards}: +44ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/boards, GET} route +1ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/boards/:id, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] BotsController {/api/bots}: +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/bots/:id, GET} route +1ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/bots, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/bots/recover, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/bots/:id/join, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/bots/:id/move, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] HighscoresController {/api/highscores}: +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/highscores/:seasonId, GET} route +1ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] RecordingsController {/api/recordings}: +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/recordings/:seasonId, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/recordings/score/last, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/recordings/:id, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] SeasonsController {/api/seasons}: +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/seasons, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/seasons/current, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/seasons/:id, GET} route +1ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/seasons/:id/rules, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] SlackController {/api/slack}: +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/slack/seasons, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/slack/season, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/slack/teams, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/slack/team, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/slack/interact, POST} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RoutesResolver] TeamsController {/api/teams}: +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [RouterExplorer] Mapped {/api/teams, GET} route +0ms
[0] [Nest] 13588 - 03/09/2024, 11:03:06 PM  LOG [NestApplication] Nest application successfully started +2ms

```

Gambar 5. Hasil jika *game engine* berhasil dijalankan

Untuk menjalankan bot, pengguna dapat memasukkan perintah “*./run-bots.bat*” jika menggunakan Windows atau “*./run-bots.sh*” jika menggunakan terminal Linux pada project directory. Jika menjalankan perintah dengan script, maka akan muncul Command Prompt sebanyak bot yang ingin dimainkan.

The image displays four separate Command Prompt windows, each showing a series of POST requests to a server endpoint. The requests are timestamped and show various moves and positions for different bots. The content of the windows is as follows:

- Top Left Window:**

```

8
>>> POST /bots/50f5e359-da22-48ab-bc91-82d53de3e456/move {'direction': 'SOUT
H'}
<<< 200 OK
9
>>> POST /bots/50f5e359-da22-48ab-bc91-82d53de3e456/move {'direction': 'EAST
'}
<<< 200 OK
8
>>> POST /bots/50f5e359-da22-48ab-bc91-82d53de3e456/move {'direction': 'WEST
'}
<<< 200 OK
9
>>> POST /bots/50f5e359-da22-48ab-bc91-82d53de3e456/move {'direction': 'WEST
'}
<<< 200 OK
8
>>> POST /bots/50f5e359-da22-48ab-bc91-82d53de3e456/move {'direction': 'WEST
'}
<<< 200 OK
|
```
- Top Right Window:**

```

Position(y=12, x=10)
diamond
>>> POST /bots/ee26688a-93c1-4634-a52e-fda9d9422e96/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=12, x=11)
diamond
>>> POST /bots/ee26688a-93c1-4634-a52e-fda9d9422e96/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=12, x=12)
diamond
>>> POST /bots/ee26688a-93c1-4634-a52e-fda9d9422e96/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=12, x=13)
diamond
>>> POST /bots/ee26688a-93c1-4634-a52e-fda9d9422e96/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=12, x=14)
diamond
>>> POST /bots/ee26688a-93c1-4634-a52e-fda9d9422e96/move {'direction': 'SOUT
H'}
<<< 200 OK
|
```
- Bottom Left Window:**

```

t', properties=Properties(points=None, pair_id=None, diamonds=None, score=None,
ne, name=None, inventory_size=None, can_tackle=None, milliseconds_left=None,
time_joined=None, base=None)
>>> POST /bots/5902741b-b281-4d8b-a787-e9cdc7b86acc/move {'direction': 'EAST
'}
<<< 200 OK
GameObject(id=1, position=Position(y=14, x=11), type='DiamondButtonGameObjec
t', properties=Properties(points=None, pair_id=None, diamonds=None, score=None,
ne, name=None, inventory_size=None, can_tackle=None, milliseconds_left=None,
time_joined=None, base=None))
>>> POST /bots/5902741b-b281-4d8b-a787-e9cdc7b86acc/move {'direction': 'SOUT
H'}
<<< 200 OK
GameObject(id=1, position=Position(y=14, x=11), type='DiamondButtonGameObjec
t', properties=Properties(points=None, pair_id=None, diamonds=None, score=None,
ne, name=None, inventory_size=None, can_tackle=None, milliseconds_left=None,
time_joined=None, base=None))
>>> POST /bots/5902741b-b281-4d8b-a787-e9cdc7b86acc/move {'direction': 'SOUT
H'}
<<< 200 OK
|
```
- Bottom Right Window:**

```

Position(y=6, x=13)
[Position(y=6, x=13), Position(y=7, x=12), Position(y=6, x=14)]
>>> POST /bots/4fc8e874-4586-44eb-884c-ad95129a7018/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=6, x=13)
[Position(y=6, x=13), Position(y=7, x=12), Position(y=6, x=14)]
>>> POST /bots/4fc8e874-4586-44eb-884c-ad95129a7018/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=6, x=13)
[Position(y=6, x=13), Position(y=7, x=12), Position(y=6, x=14)]
>>> POST /bots/4fc8e874-4586-44eb-884c-ad95129a7018/move {'direction': 'EAST
'}
<<< 200 OK
Position(y=6, x=13)
[Position(y=6, x=13), Position(y=7, x=12), Position(y=6, x=14)]
>>> POST /bots/4fc8e874-4586-44eb-884c-ad95129a7018/move {'direction': 'EAST
'}
<<< 200 OK
|
```

Gambar 6. Command Prompt yang terbuka sebanyak jumlah bot yang dijalankan pada script

## BAB III

### APLIKASI STRATEGY GREEDY

#### 3.1. Alternatif Greedy

##### 3.1.1. Greedy By Distance

Pada pendekatan ini, target yang dipilih akan berdasarkan jarak minimum antara bot ke target. Perhitungan jarak terdekat ini juga mempertimbangkan keberadaan teleporter yang mungkin dapat memotong jarak dari bot ke target.

- **Mapping Elemen Greedy**

- Himpunan kandidat: kumpulan diamond, base, atau teleporter
- Himpunan solusi: diamond, base, atau teleporter yang terpilih
- Fungsi seleksi: Jarak ke target harus minimum, hal tersebut didapat dengan membandingkan jarak bot ke target secara langsung dan jarak bot ke target via teleporter.
- Fungsi kelayakan: Memeriksa apakah jumlah diamond dalam inventory sudah sama dengan 5. Selain itu, terdapat penanganan edge case dimana kita akan menghindari diamond merah ketika inventory kita sudah berisi 4 diamond. Selain itu, edge case yang ditangani adalah jarak antara portal masuk tidak boleh sama dengan player bila ingin mencari portal masuk.
- Fungsi obyektif: Memaksimalkan jumlah diamond yang didapat.

- **Analisis Efisiensi Solusi**

Fungsi `next_move` pada pendekatan ini berawal dengan penentuan portal mana yang merupakan portal masuk dan portal mana yang merupakan portal keluar. Hal tersebut dilakukan dengan cara mencari jarak minimal antara bot dengan salah satu portal, dimana porta dengan jarak terkecil dari bot akan menjadi portal masuk. Prosedur tersebut memiliki efisiensi  $O(1)$  dilanjut dengan penentuan apakah inventory sudah penuh. Berarti pada kondisi tersebut, algoritma ini memiliki efisiensi  $O(1)$ . Namun pada kondisi yang normal, yaitu bot masih ingin mencari diamond, efisiensinya adalah  $O(n)$  dimana  $n$  adalah jumlah diamond. Hal tersebut terjadi saat penentuan jarak portal ke diamond serta penentuan jarak portal ke player.

- **Analisis Efektifitas Solusi**

Strategi ini efektif bila:

- Bot berada pada daerah dengan densitas objek yang cukup tinggi.
- Bot berada pada board yang tidak begitu dinamis. Board yang dinamis dapat membuat target yang ingin dicapai bot tidak begitu konsisten. Akibatnya banyak langkah yang hambur untuk mengganti tujuan.
- Jarak antar teleporter jauh.
- Target dekat dengan teleporter.

### 3.1.2. Greedy By Points

Pada pendekatan ini, bot akan lebih mengutamakan diamond merah, dengan kata lain diamond dengan poin tertinggi. Walau pada dasarnya algoritmanya adalah seperti itu, bila diamond merah sudah habis maka akan pindah ke shortest distance.

- **Mapping Elemen Greedy**

- Himpunan kandidat: kumpulan diamond, base, atau teleporter
- Himpunan solusi: diamond, base, atau teleporter yang terpilih
- Fungsi seleksi: ada tidaknya diamond merah pada board. Bila ada, maka utamakan ke diamond merah.
- Fungsi kelayakan: Memeriksa apakah jumlah diamond dalam inventory ditambah diamond yang dipilih tidak melebihi 5. Selain itu, terdapat penanganan edge case dimana kita akan menghindari diamond merah ketika inventory kita sudah berisi 4 diamond.
- Fungsi obyektif: Memaksimalkan jumlah diamond yang didapat.

- **Analisis Efisiensi Solusi**

Untuk kasus di mana inventory sudah penuh, maka efisiensi algoritma ini adalah  $O(1)$  karena algoritmanya sama dengan penanganan inventory penuh pada algoritma greedy by points. Bila masih ada ruangan di inventory maka bot akan mencari diamond merah yang terdekat ,bila ada. Untuk mencari diamond distance yang terdekat, maka algoritma akan mengiterasi dari tiap diamond merah yang ada pada board dan menemukan nilai minimalnya. Kedua hal tersebut mempunyai

efisiensi  $O(n)$  dengan  $n$  adalah banyak diamond merah. Bila diamond merah sudah tidak ada, maka kita akan mencari diamond terdekat, hal tersebut juga  $O(n)$ .

- **Analisis Efektifitas Solusi**

Strategi ini efektif bila:

- Ada, dan lebih baiknya lagi , banyak diamond merah pada board.
- Terdapat diamond biru yang terletak di path ke diamond merah
- Bot berada pada posisi dengan densitas diamond tinggi.

### 3.1.3. Greedy By Point Density

Pada pendekatan ini, target yang dipilih merupakan target dengan densitas terbaik dari semua pilihan diamond. Densitas yang dimaksud disini merupakan points yang dimiliki diamond (2 atau 1) dibagi dengan jarak dari bot menuju ke diamond tersebut. Jarak yang dimaksud dapat berarti jarak langsung dari bot menuju diamond atau melalui teleporter.

- **Mapping Elemen Greedy**

- Himpunan kandidat: Kumpulan diamond yang mungkin diambil
- Himpunan solusi: Urutan mengambil diamonds
- Fungsi seleksi: Pilihlah diamond dengan densitas terbesar. yang didapat dengan membagi point diamond dengan total langkah yang dibutuhkan untuk mencapai diamond tersebut dari posisi bot, enta dengan atau tanpa melalui bot.
- Fungsi kelayakan: Memeriksa apakah jumlah diamond yang dipilih bila dijumlah dengan diamond dalam inventory tidak melebihi 5.
- Fungsi obyektif: Memaksimalkan jumlah diamond yang didapat.

- **Analisis Efisiensi Solusi**

Sebelum menjalankan algoritma greedy, kita perlu mengecek beberapa kondisi untuk memastikan keberlangsungan terbaik di permainan ini .

Strategi dimulai dengan mengecek apakah diamond yang dimiliki oleh bot saat ini berjumlah 4. Pengecekan ini krusial untuk memastikan tidak adanya error yang disebabkan oleh pembagian dengan 0 apabila nanti diamond dengan density terbaik ternyata merupakan diamond merah. Kasus ini dapat terjadi karena ketika kita sampai pada diamond berwarna merah namun poin 4, diamond tidak akan

terlambil. Otomatis, diamond merah menjadi himpunan kandidat yang akan dievaluasi dan terjadilah pembagian dengan 0.

Selanjutnya, dicek apakah jumlah diamond yang ada di inventory sudah berukuran 5 atau sisa waktu permainan kurang dari 7 detik. Jika iya, maka bot harus langsung kembali ke base untuk mengembalikan semua diamond dan mengosongkan *inventory*.

Selanjutnya, dicek apakah sisa diamond di permainan kurang dari 3. Jika iya, maka bot akan diarahkan untuk menuju ke red button dan mereset *state* permainan agar lebih banyak diamond yang bisa dikumpulkan.

Jika semua kondisi khusus sebelumnya tidak ada yang terpenuhi, algoritma greedy aman untuk dijalankan. Algoritma ini dimulai dengan menginisiasi *density* maksimal dengan 0 serta position dengan elemen pertama dari list diamond. Selanjutnya, dilakukan iterasi ke seluruh elemen dari list. Di setiap iterasi, dilakukan perhitungan densitas dengan membagi point diamond dengan jaraknya ke bot terkait. Lalu, dilakukan pengecekan apakah densitas terkait lebih baik dari densitas maksimal terakhir. Jika iya, ubah densitas maksimal ke variabel tersebut serta masukkan position diamond pula. Jika semua elemen sudah teriterasi, kembalikan posisi densitas maksimal.

Secara umum algoritma ini memiliki kompleksitas waktu  $O(n)$  dengan  $n$  adalah banyak data dalam *list diamond*.

#### • **Analisis Efektifitas Solusi**

Strategi ini efektif apabila kondisi berikut terpenuhi:

- Diamonds tersebar secara merata dan relatif tidak jauh dari posisi bot saat ini.
  - Tidak terlalu banyak pemain yang mengikuti permainan
  - Terdapat cukup banyak *red diamonds* pada permainan
- Strategi ini lemah apabila terjadi beberapa kondisi berikut:
- Pilihan *diamonds* yang tersedia berada jauh dari jangkauan.

Hal ini menjadi sebuah masalah karena semakin jauh jarak suatu diamond dari bot, maka toleransi akan jarak tersebut menjadi semakin besar. Perhatikan perbandingan berikut :

Tabel I Perbandingan Densitas untuk Variasi Diamond dan Langkah

Densitas	Langkah yang dibutuhkan untuk sampai	
	Blue Diamond	Red Diamond
1/2	2	4
1/3	3	6
1/4	4	8
1/5	5	10
1/x	x	2x

Untuk kasus dengan jarak yang kecil (misal dibawah 3) perbandingan ini masih bisa ditoleransi dengan baik. Misalkan alih-alih mengambil *blue diamond* yang berjarak 3 dari bot, kita lebih baik mengambil *red diamond* yang berjarak 5. Namun dalam skenario *blue diamond* dengan jarak 5 dibandingkan dengan *red diamond* berjarak 9, hal ini menjadi kurang optimum. Hal ini disebabkan karena sudah terlalu besar *gap* antara point dengan jarak yang menjadi perbandingan. Bot malah akan *ignore* sesuatu yang 4 steps lebih dekat dari

- Kurang banyaknya *red diamonds* dalam permainan

Konsiderasi akan *red diamonds* merupakan keunggulan utama dari strategi ini. Tanpa adanya banyak *red diamonds* tersebut, strategi ini menjadi sama saja dengan mencari diamonds dengan jarak terdekat dari bot.

### 3.1.4. Greedy By Area Density

Pada pendekatan ini, bot akan melakukan perhitungan dengan mencari jumlah diamond di sekitar tiap diamond. Pencarian akan dilakukan pada kotak 3x3 di sekitar diamond. Jumlah poin diamond pada kotak akan dibagi dengan jarak pada diamond pusat pilihan yang menjadi acuan pencarian diamond pada area.

- **Mapping Elemen Greedy**

- Himpunan kandidat: kumpulan diamond atau base
- Himpunan solusi: diamond atau base yang terpilih dari himpunan kandidat

- Fungsi seleksi: area dengan nilai poin dibagi jarak ke pusat area terbesar. Area didapatkan dengan mencari diamond pada kotak 3x3 dengan setiap diamond sebagai pusatnya.
- Fungsi kelayakan: memeriksa apakah jumlah diamond sudah 5 (atau sudah 4 dalam kasus diamond incaran selanjutnya memiliki poin 2). Selain itu, mempertimbangkan apakah bot sedang dalam posisi dapat di-tackle atau tidak. Jika ya, maka yang menjadi langkah selanjutnya adalah menghindar, tanpa mengubah himpunan solusi.
- Fungsi obyektif: Memaksimalkan jumlah diamond yang didapat.

#### ● **Analisis Efisiensi Solusi**

Terdapat beberapa method dari kelas SquareDensity yang akan dipanggil setiap kali method next\_move dipanggil, yaitu \_\_escape\_from\_enemy dan \_\_max\_area\_to\_distance. Method \_\_escape\_from\_enemy bertujuan untuk mengembalikan posisi untuk dituju jika di sekitar bot ada bot lain untuk menghindari tabrakan. Kompleksitas algoritma ini untuk m bot adalah  $O(m)$ . Sedangkan kompleksitas algoritma \_\_max\_area\_to\_distance untuk n diamonds adalah  $O(n^2)$ . Demikian pula kompleksitas untuk next\_move adalah  $O(n^2)$  karena didominasi oleh \_\_max\_area\_to\_distance.

Solusi algoritma ini efisien bila banyak diamonds yang berdekatan, dikarenakan jika tidak, komputasi yang memakan waktu lebih lama ( $O(n^2)$  alih-alih  $O(n)$  jika menggunakan strategi yang sebelumnya) hanya akan menghasilkan hasil yang sama karena luas kotak yang digunakan sebatas 3x3.

#### ● **Analisis Efektifitas Solusi**

Strategi ini efektif bila:

- Mekanisme tackle dinonaktifkan, karena akan ada kemungkinan bot tidak melakukan komputasi ulang jika berpindah lokasi karena melalui portal atau kembali ke base akibat tackle oleh pemain lawan.
- Jumlah pemain lawan sedikit sehingga diamond-diamond yang sudah dipetakan ada pada area tertentu tidak mudah hilang dari path yang akan meningkatkan kemungkinan komputasi density diulang kembali.

### 3.2. Strategi Greedy yang Diimplementasikan

Berdasarkan hasil ujicoba yang telah dilakukan terhadap keempat alternatif bot, tim kami memutuskan untuk mengimplementasikan bot dengan algoritma *Greedy by Point Density*. Keputusan ini didasarkan pada minimnya kasus di mana *Greedy by Point Density* tidak berfungsi dengan baik. Kasus-kasus ini terutama terbatas pada kondisi di mana diamond merah berada sangat jauh dari bot, tetapi tidak cukup jauh untuk membuat bot tersebut mengambil diamond biasa yang lebih dekat. Algoritma ini menunjukkan kinerja yang lebih baik dalam hal ini dibandingkan dengan algoritma-algoritma lain.

Berikut adalah ringkasan kelemahan dari setiap alternatif bot lainnya:

- Square Density: Tidak efektif ketika musuh tersebar luas dan diamonds dekat dengan mereka. Hal ini membuat bot Square Density kurang ideal karena cenderung tidak mampu mengoptimalkan pengambilan diamond.
- Value Density: Efektif hanya ketika ada banyak diamond merah di papan permainan. Bot ini mungkin tidak memberikan hasil terbaik jika diamond merah yang tersedia terbatas.
- Jarak Terdekat: Memilih jarak terpendek tanpa mempertimbangkan nilai diamond. Misalnya, bot ini mungkin akan memilih diamond biru yang jaraknya lebih dekat daripada diamond merah yang memiliki nilai lebih tinggi tetapi jaraknya hanya 1-2 langkah lebih jauh.

Dengan mempertimbangkan kelemahan dan keunggulan masing-masing algoritma, tim kami memilih *Greedy by Point Density* karena kemampuannya yang lebih baik dalam menangani kasus-kasus umum dalam permainan diamonds. Kemampuan greedy ini untuk lebih konsisten dalam jumlah percobaan yang panjang, membuat kami memilih metode ini.

## BAB IV

# IMPLEMENTASI DAN PENGUJIAN

### 4.1. Implementasi dalam *Pseudocode*

#### Greedy by Distance

```

Class TeleportGreed inherits BaseLogic:
    Constructor:
        Initialize goal_position as None
        Initialize current_direction as 0

    Method next_move(board_bot: GameObject, board: Board) -> (delta_x, delta_y):
        list_teleporters <- [d for d in board.game_objects if d.type ==
        "TeleportGameObject"]
        marco <- list_teleporters[0]
        polo <- list_teleporters[1]
        props <- board_bot.properties
        current_position <- board_bot.position
        target_teleporter <- marco
        exit_teleporter <- polo
        dm <- manhattan distance between marco.position and current_position
        dp <- manhattan distance between polo.position and current_position
        if dp < dm then
            target_teleporter <- polo
            exit_teleporter <- marco

        {Analyze new state}
        if props.diamonds == 5 then
            base <- board_bot.properties.base
            teleporter_base_distance <- manhattan distance between
            exit_teleporter.position and base
            distance_to_target_teleporter <- manhattan distance between
            target_teleporter.position and base
            td <- teleporter_base_distance + distance_to_target_teleporter
            distance_bot_base <- manhattan distance between base and
            current_position
            if td < distance_bot_base then
                Calculate step (delta_x and delta_y) to target_teleporter
                return delta_x, delta_y
            else:
                goal_position <- base
        else:
            {Will later enter gather diamond phase}
            goal_position <- None

        {Print current_position}
        if goal_position is not None then:
            Calculate direction to goal_position
            return delta_x, delta_y
        else:
            Find the closest diamond to the player
            Calculate distances between diamonds and teleporters
    
```

```

Calculate total teleporter distance
if total_teleporter_distance < minimum_diamond_distance then:
    Go to the teleporter
else:
    Find the closest diamond
    Calculate direction to chosen diamond

return delta_x, delta_y

```

*Greedy by Points*

```

class points inherits BaseLogic
Constructor:
    Initialize goal_position as None
    Initialize current_direction as 0

Method next_move (board_bot:GameObject, board: board) -> tuple
    list_teleporters <- array gameObject dengan tipe = "TeleportGameObject"
    marco <- list_teleporters[0]
    polo <- list_teleporters[1]
    props <- board_bot.properties
    current_position <- board_bot.position
    targetTeleporter, exitTeleporter <- determineTeleporter(marco, polo)
    {menentukan mana telporter masuk dan teleporter keluar}
    if props.diamonds == 5 then
        td <- jarak ke base bila lewat teleporter
        distanceBotBase <- jarak ke base bila secara langsung
        if td < distanceBotBase then
            (delta_x,delta_y) <-
            get_direction(current_position.x,current_position.y,targetTeleporter.position.
x,targetTeleporter.position.y)
            -> (delta_x,delta_y)
        else
            goal <- base
        if goal:
            (delta_x,delta_y) <-
            get_direction(current_position.x,current_position.y,targetTeleporter.position.
x,targetTeleporter.position.y)
        else
            redDiamonds <- array semua diamond merah
            diamondDistance <- cari jarak2 dari diamond merah ke bot lalu taruh
            ke array ini
            diamods <- board.diamonds
            allDiamondDistance <- cari jarak2 semua diamond ke bot dan masukkan ke
            array ini
            if props.diamonds == 4 then
                chosenDiamond <- diamond biru yang terdekat
            else
                if tidak ada red diamond then
                    chosenDiamond <- diamond biru yang terdekat
                else
                    chosenDiamond <- diaamond merah terdekat
                    (delta_x,delta_y) <-
                    get_direction(current_position.x,current_position.y,chosenDiamond.position.x,c

```

```

hosenDiamond.position.y)
-> (delta_x,delta_y)

```

### Greedy by Point Density

```

Class PureDensityBot inherits BaseLogic:
    Method __init__():
        Initialize goal_position as None
        Initialize current_direction as 0

    Method needed_steps(starting_pos: Position, diamond_pos: Position) -> int:
        {Calculate the number of steps needed to reach the diamond_pos from starting_pos}
        return the manhattan distance between starting_pos and diamond_pos

    Method get_density(diamond: GameObject, bot_pos: Position) -> float:
        {Calculate the density of a diamond based on its points and distance from the bot}
        return diamond.properties.points / self.needed_steps(bot_pos,
diamond.position)

    Method generate_best_density(diamonds: List[GameObject], board_bot:
GameObject, start_teleporter_position: Position, end_teleporter_position:
Position, distance_to_targetTeleporter : int) -> None:
        {Find the diamond with the highest density and set it as the goal position.}
        curr_density_max <- 0
        curr_density_max_pos <- diamonds[0].position
        bot_position <- board_bot.position

        {calculate density of each diamond and compare it with the current max density}
        for diamond in diamonds:
            density <- self.get_density(diamond, bot_position)
            {Calculate density based on the distance to the teleporter}
            teleportdensity <- diamond.properties.points /
(distance_to_targetTeleporter + self.needed_steps(diamond.position,
end_teleporter_position) )
            if density > curr_density_max or teleportdensity >
curr_density_max:
                if (teleportdensity > density):
                    curr_density_max <- teleportdensity
                    curr_density_max_pos <- start_teleporter_position
                else:
                    curr_density_max <- density
                    curr_density_max_pos <- diamond.position
            {Langsung set as goal}
            self.goal_position = curr_density_max_pos

```

```

Method generate_shortest_blue(diamonds: List[GameObject], board_bot: GameObject, diamond_button: GameObject) -> None:
    diamonds.sort(key=lambda x: self.needed_steps(board_bot.position, x.position))
    for diamond in diamonds:
        if diamond.properties.points = 1 then
            self.goal_position <- diamond.position
            break
        else:
            self.goal_position <- diamonds[0].position

Method next_move(board_bot: GameObject, board: Board) -> tuple:
    {initialize props}
    props <- board_bot.properties
    base <- board_bot.properties.base
    list_diamonds <- board.diamonds
    current_position <- board_bot.position

    {initialize teleporters}
    list_teleporters <- [d for d in board.game_objects if d.type == "TeleportGameObject"]
    marco <- list_teleporters[0]
    polo <- list_teleporters[1]
    targetTeleporter <- marco
    exitTeleporter <- polo
    dm <- self.needed_steps(marco.position, current_position)
    dp <- self.needed_steps(polo.position, current_position)

    diamond_button <- [d for d in board.game_objects if d.type == "DiamondGameObject"]
    {Cari teleporter terdekat}
    if dp < dm then:
        targetTeleporter = polo
        exitTeleporter=marco

    {Cari jarak ke teleporter terdekat}
    distance_to_targetTeleporter <-
    self.needed_steps(targetTeleporter.position, current_position)
    {Kalau Sudah 4, pastikan cari biru terdekat}

    if props.diamonds = 4 then:
        self.generate_shortest_blue(list_diamonds, board_bot,
        diamond_button[0].position)
        {If there are 5 diamonds in the inventory, move towards the base}
        else if props.diamonds = 5 or (props.milliseconds_left < (1000*
        self.needed_steps(base, board_bot.position)+2000) and props.milliseconds_left
        < 8000):
            teleporter_base_distance <-
            self.needed_steps(exitTeleporter.position,base)
            td <- teleporter_base_distance + distance_to_targetTeleporter
            distanceBotBase <- self.needed_steps(base,current_position)
            if (td < distanceBotBase) and distance_to_targetTeleporter <> 0
    then:
        delta_x, delta_y <- get_direction(
            current_position.x,
            current_position.y,

```

```

        targetTeleporter.position.x,
        targetTeleporter.position.y,
    )
    return delta_x,delta_y
else:
    self.goal_position = base
{If there is only some diamonds left and the distance of our target
is less than the distance to the diamond button, move towards the diamond
trigger button}
else if len(list_diamonds) < 3 and self.goal_position!=None and
self.needed_steps(board_bot.position, self.goal_position) >
self.needed_steps(diamond_button[0].position, board_bot.position) then:
    self.goal_position = diamond_button[0].position
{If the goal position is not among the diamonds or it's not set yet,
generate the best density}
{If targetTeleporter is not among the teleporters or it's not set
yet, generate the best density}
else if all(self.goal_position != diamond.position for diamond in
list_diamonds) or all (targetTeleporter.position != teleporter.position for
teleporter in list_teleporters) or self.goal_position is None:
    self.generate_best_density(list_diamonds, board_bot,
targetTeleporter.position, exitTeleporter.position,
distance_to_targetTeleporter)
    current_position <- board_bot.position
    delta_x, delta_y <- get_direction(current_position.x,
current_position.y, self.goal_position.x, self.goal_position.y)

return delta_x, delta_y

```

*Greedy by Area Density*

```

Class SquareDensity inherits Baselogic:
Constructor:
    goal_position <- None {Optional[Position]}
    target_path <- [] {List[GameObject]}
    achieved_head <- False {Optional[bool]}
    path_head <- None {Optional[GameObject]}

Method __escape_from_enemy(my_bot: GameObject, board: Board) -> Position:
    list_bots <- board.bots
    for each a_bot in list_bots:
        if a_bot.properties.name <> my_bot.properties.name then:
            distance_x <- absolute difference between a_bot.x and my_bot.x
            distance_y <- absolute difference between a_bot.y and my_bot.y
            if distance_x = 1 and distance_y = 0 then:
                return Position(x=my_bot.position.x,
y=abs(my_bot.position.y-1)
                else if distance_y = 1 and distance_x = 0 then:
                    return Position(x=abs(my_bot.position.x-1),
y=my_bot.position.y)
                else: {return bot's own position if no nearby bots}
                    return Position(x=my_bot.position.x, y=my_bot.position.y)

```

```

Method __max_area_to_distance(my_bot: GameObject, board: Board):
    self.achieved_head <- False
    list_diamond <- board.diamonds
    length <- Length of list_diamond
    density <- 0
    {iterate through every diamond}
    for i in range from 0 to length:
        count <- 1
        temp_diamonds <- [list_diamond[i]]
        curr_diamond <- list_diamond[i]
        for j in range from 0 to length:
            {iterate every diamond pairs except when paired w/ itself}
            if i <> j and is in 3 by 3 square around curr_diamond then:
                {preventing freeze due to bot with 4 diamonds in inventory
trying to get to red diamond}
                if my_bot.properties.diamonds <> 4 or
                list_diamond[j].properties.points <> 2 then:
                    temp_diamonds.append(list_diamond[j])
                    count++
                    distance <- manhattan distance between temp_diamonds[0].position and
my_bot.position
                    if distance <> 0 then:
                        temp_density <- count / (distance * 5)
                        if temp_density > density then:
                            density M <- temp_density
                            target_path <- temp_diamonds
                            path_head <- target_path[0]

```

```

Method next_move(my_bot: GameObject, board: Board):
    props <- Properties of my_bot
    escape_position <- __escape_from_enemy(my_bot, board)

    {if no bot nearby}
    if escape_position = my_bot.position then:
        {go to base whatever the condition if inventory is full}
        if my_bot's diamonds count = 5 then:
            self.goal_position <- props.base
        Else:
            temp <- []
            for each x in self.target_path:
                if x is in board.diamonds then:
                    temp.append(x)
            self.target_path <- temp

            if len(self.target_path) <> 0 then:
                if props.diamonds = 4 and self.target_path[0].points = 2:
                    Remove the first element from self.target_path
                else if achieved_head is False and path_head is not in
self.target_path then:
                    __max_area_to_distance(my_bot, board)
                else if achieved_head is True then:
                    if my_bot.position = self.target_path[0].position then:
                        Remove the first element from self.target_path
                    else if my_bot.position = path_head.position then:
                        self.achieved_head <- True
                        Remove the first element from self.target_path

```

```

    if len(self.target_path) = 0 then:
        __max_area_to_distance(my_bot, board)
        self.goal_position <- self.target_path[0].position
{there is a nearby bot, therefore, RUN!}
else:
    self.goal_position <- escape_position

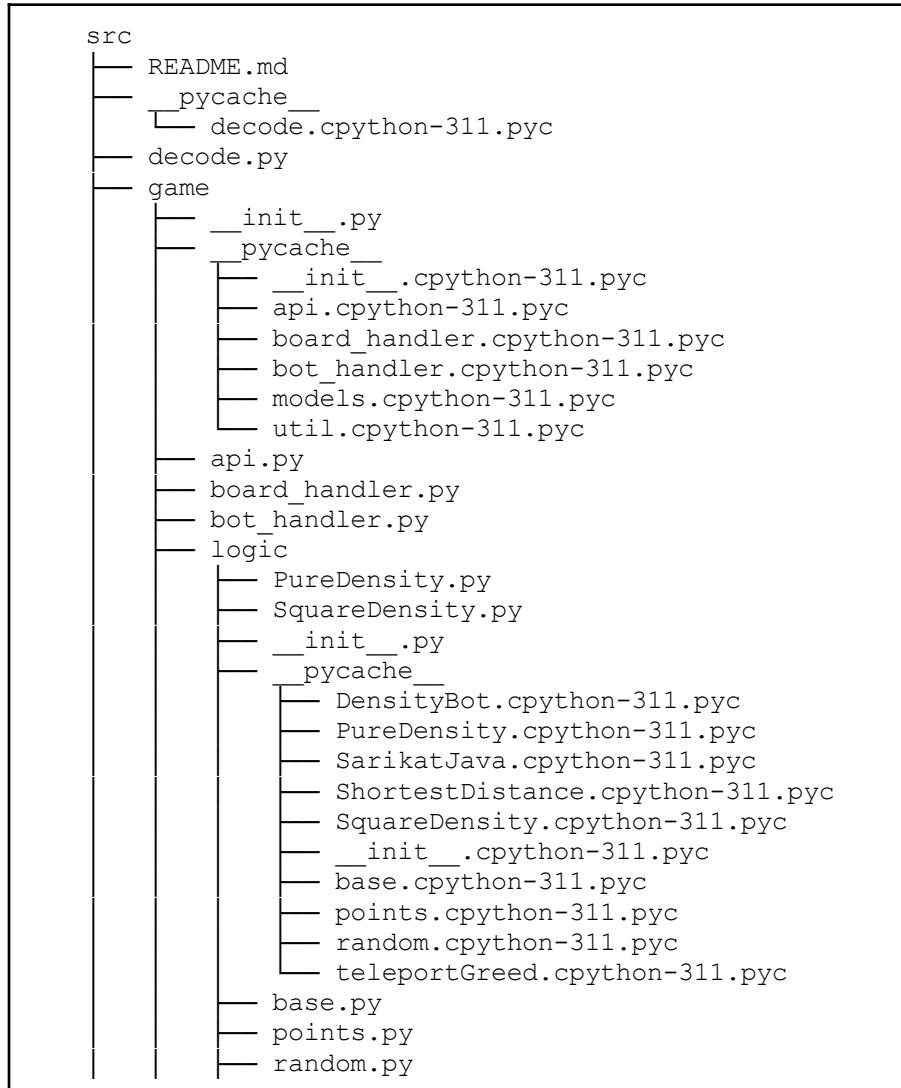
delta_x, delta_y <- get_direction(current_position.x, current_position.y,
self.goal_position.x, self.goal_position.y)

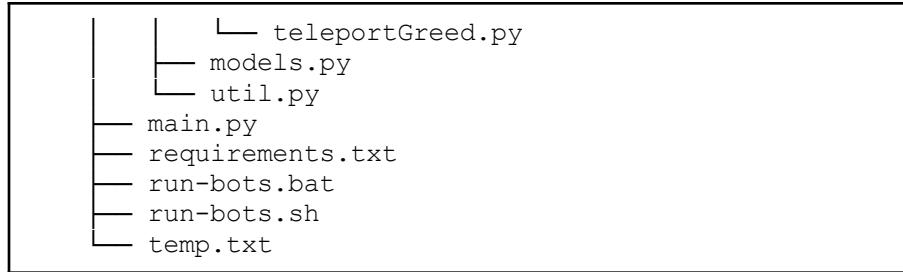
return delta_x, delta_y

```

## 4.2. Struktur Data Program

Secara keseluruhan, struktur program bot Etimo: Diamond yang kami buat adalah sebagai berikut:





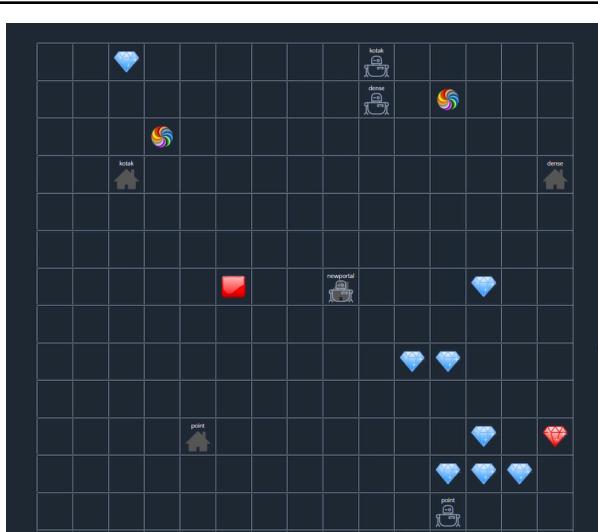
Seluruh implementasi logic bot diletakkan pada directory `src/game/logic`. Logic yang dibuat diimplementasikan dengan menciptakan suatu kelas yang menurunkan (*inherit*) kelas *BaseLogic*. Implementasi kelas ini wajib memiliki *method* `next_move` yang akan dipanggil pada setiap giliran pada `src/main.py`. Secara umum bot yang diimplementasikan memerlukan informasi keadaan board saat itu. Karena informasi yang diberikan, *method* `next_move` dapat mempertimbangkan berbagai hal, seperti lokasi objek-objek pada board serta kapasitas dan isi *inventory*.

### 4.3. Analisis dan Pengujian

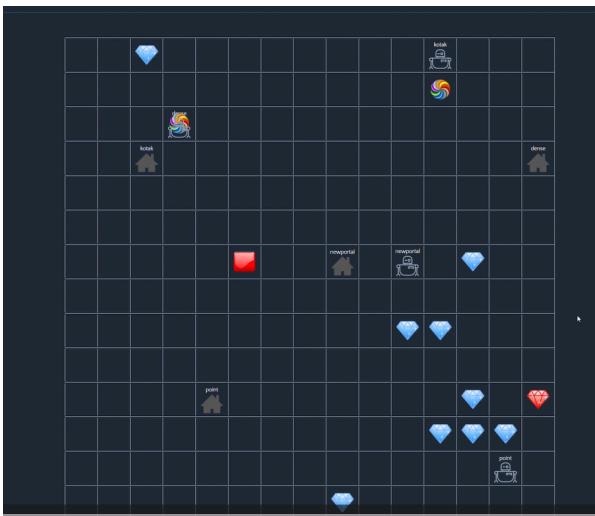
Tabel II Statistik dari Empat Bot yang Diuji

Round	Area Density	Density	Distance	Point
1	9	10	10	10
2	15	6	10	0
3	0	8	1	8
4	10	6	9	5
5	13	8	10	10
6	5	10	8	5
7	10	11	10	0
8	10	8	10	0
9	10	9	12	12
10	5	10	5	5
<b>Average</b>	8.7	8.6	8.5	5.5
<b>STDEV</b>	4.321779469	1.7127	3.205897	4.478343
<b>Total Win</b>	4	5	3	3

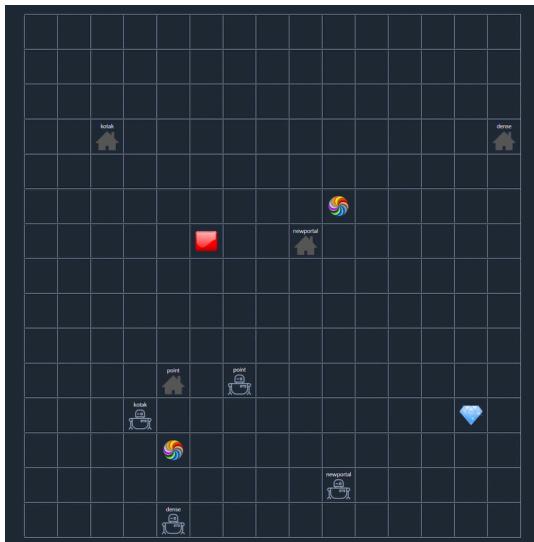
Dari hasil pengujian terhadap keempat bot, terlihat bahwa Area Density memperoleh rata-rata jumlah diamond terbanyak dibandingkan dengan ketiga bot lainnya, meskipun perbedaannya hanya 1 diamond secara keseluruhan dibanding Density. Namun, perlu diperhatikan bahwa bot Density mencatat total kemenangan paling banyak dan memiliki standar deviasi yang paling rendah. Hal ini menunjukkan bahwa bot Density memberikan probabilitas kemenangan yang paling baik dan konsisten dibandingkan ketiga bot lainnya. Sementara Area Density, saat berada dalam situasi di mana banyak diamond terkumpul di suatu tempat, akan dominan. Namun, di luar situasi tersebut, kinerjanya menjadi tidak konsisten. Oleh karena itu, bot Density dapat dianggap sebagai pilihan terbaik untuk digunakan, karena memberikan kemenangan yang baik dan konsisten dalam berbagai situasi.



Dari gambar terkait, dapat dilihat bahwa bot “dense” memasuki teleporter untuk mendapatkan rute terpendek ke 1 buah diamond yang ada di ujung kiri atas. Hal ini menunjukkan bahwa solusi yang diberikan **OPTIMUM**



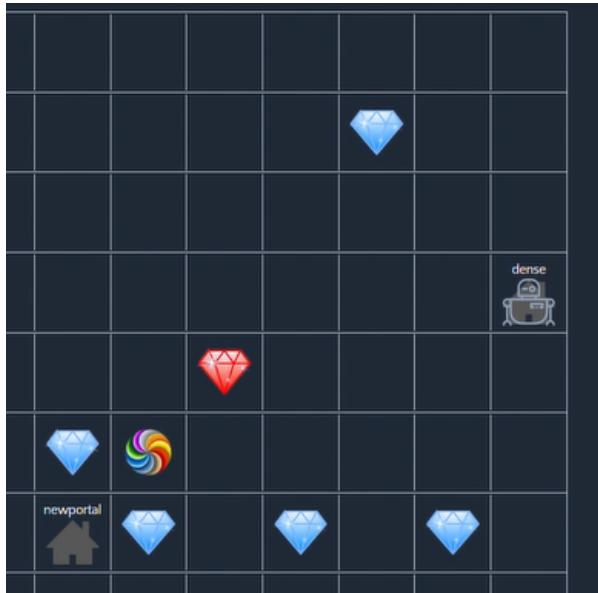
**Pengujian 1.** Mencari jarak terpendek ke diamond  
dengan memanfaatkan TELEPORTER



Dari gambar terkait, dapat dilihat bahwa bot “dense” ketika sudah memiliki banyak diamond == 5, akan langsung menuju ke base dan menggunakan teleporter untuk mempercepat proses sampai. Untuk kasus ini, hasil yang diberikan **OPTIMUM**.



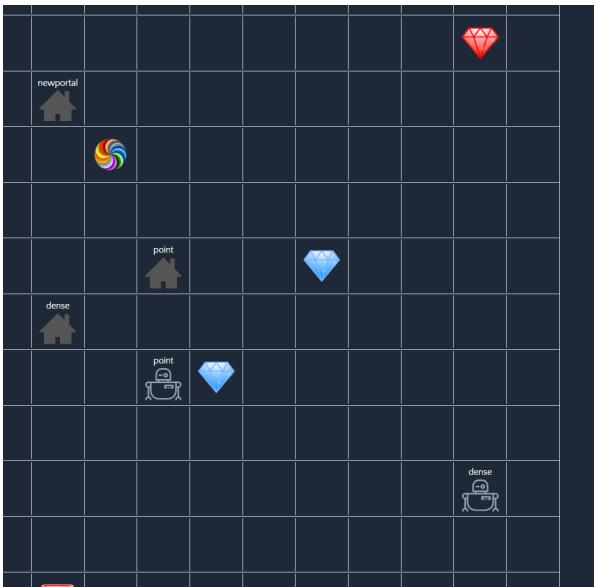
Pengujian 2. Kasus diamond == 5



Dari gambar terkait, dapat dilihat bahwa bot “dense” mengambil diamond merah dengan jarak yang sedikit lebih jauh dibandingkan diamond biru paling dekat karena densitasnya lebih besar. Dari sini, dapat disimpulkan hasil yang didapat **OPTIMUM**.



**Pengujian 3.** Kasus diamond merah dengan jarak 5, dan ada 2 diamond biru dengan jarak 4



**Pengujian 4.** Kasus dimana diamond merah terletak sangat jauh

Dari gambar terkait, dapat dilihat bahwa bot menuju ke arah diamond merah yang sangat jauh, walaupun di dekatnya terdapat 2 diamonds yang setara. Ini merupakan salah satu kelemahan dari algoritma ini dan menyebabkan hasil **TIDAK OPTIMUM**.

## BAB V

# SIMPULAN DAN SARAN

### 5.1. Simpulan

Dalam pelaksanaan Tugas Besar I IF2211 Strategi Algoritma, kami telah melakukan eksplorasi secara mendalam untuk mengimplementasikan logika bot pada permainan Etimo: Diamonds dengan menggunakan pendekatan Greedy. Kami memilih solusi yang menggabungkan berbagai elemen yang telah kami uji dan eksplorasi selama tahap eksperimen kami, yaitu bot Greedy By Point Density.

Dalam proses eksplorasi, kami menyadari bahwa algoritma Greedy mungkin tidak sepenuhnya cocok untuk digunakan dalam permainan ini. Hal ini disebabkan oleh sifat permainan yang sangat dinamis dan adanya faktor luck yang cukup besar. Meskipun algoritma Greedy cenderung memilih langkah yang secara lokal optimal pada setiap langkahnya, strategi ini mungkin tidak dapat mengakomodasi dengan baik dinamika permainan yang kompleks dan keberuntungan yang signifikan dalam penempatan diamond.

Oleh karena itu, kami memilih untuk menggabungkan pendekatan Greedy dengan fokus pada *Point Density*. Dengan demikian, bot kami memiliki kemampuan untuk mempertimbangkan langkah yang paling baik di situasi yang berbeda-beda dengan berbagai konsiderasi tambahan, meskipun masih mempertahankan unsur Greedy untuk memaksimalkan keuntungan pada setiap langkah.

Kami percaya bahwa pendekatan ini dapat memberikan keseimbangan yang baik antara kecepatan dalam pengambilan keputusan (yang merupakan keunggulan dari algoritma Greedy) dan kemampuan untuk menanggapi secara efektif terhadap dinamika permainan dan faktor luck yang signifikan.

### 5.2. Saran

1. Membuat timeline kerja yang dapat dijadikan acuan agar kemajuan kerja lebih terukur.
2. Lebih mempertimbangkan bahwa dalam beberapa permasalahan, pendekatan *Greedy* bukanlah pendekatan yang paling optimal dan tidak selalu dapat memberikan solusi optimal.

3. Mencari permainan yang lebih sedikit menggunakan faktor *luck* didalamnya, jika memang ingin menggunakan algoritma greedy sebagai solusi.

## LAMPIRAN

### Repositori Github

[https://github.com/ImmanuelSG/Tubes1\\_SarikatJava.git](https://github.com/ImmanuelSG/Tubes1_SarikatJava.git)

### Link Video Tugas Besar I Strategi Algoritma

<https://youtu.be/iaNjdIVSZvs?si=-pZ8nnXjOxGT29mK>

### Screenshot Hasil Testing

Season rules	
Final Score	
Name	Score
dense	10
point	10
newportal	10
square	9

Final Score	
Name	Score
square	15
newportal	10
dense	6
point	0

Final Score	
Name	Score
dense	10
point	8
newportal	1
square	0

Final Score	
Name	Score
square	10
newportal	9
dense	6
point	5

Final Score	
Name	Score
square	13
newportal	10
point	10
dense	8

Final Score	
Name	Score
dense	10
newportal	8
point	5
square	5

Final Score	
Name	Score
dense	11
square	10
newportal	10
point	0

Final Score	
Name	Score
newportal	10
square	10
dense	8
point	0

Final Score	
Name	Score
point	12
newportal	12
square	10
dense	9

Final Score	
Name	Score
dense	10
newportal	5
square	5
point	5

## DAFTAR PUSTAKA

- E. Demaine, “Greedy Algorithms: Minimum Spanning Tree,” 6.046J Design and Analysis of Algorithms, 2015. [Online]. Available: [https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/mit6\\_046js15\\_lec12/](https://ocw.mit.edu/courses/6-046j-design-and-analysis-of-algorithms-spring-2015/resources/mit6_046js15_lec12/). [Accessed: Mar. 9, 2024]
- R. Munir, “Algoritma Greedy (Bagian 1),” IF2211 Strategi Algoritma, 2024. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf). [Accessed: Mar. 9, 2024].