

# LAPORAN TUGAS BESAR 1 IF2211

## STRATEGI ALGORITMA

*PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN  
DIAMONDS*



**Dosen Pengampu** : Dr. Nur Ulfa Maulidevi, S. T, M.Sc  
**Asisten pembimbing** : Sulthan Dzaky Alfaro

**Disusun oleh:**  
**Kelas 02 - Kelompok 15 - blinkBlink**

**Angelica Kierra Ninta Gurning (13522048)**  
**Marzuli Suhada M (13522070)**  
**Muhammad Neo Cicero Koda (13522108)**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**2024**

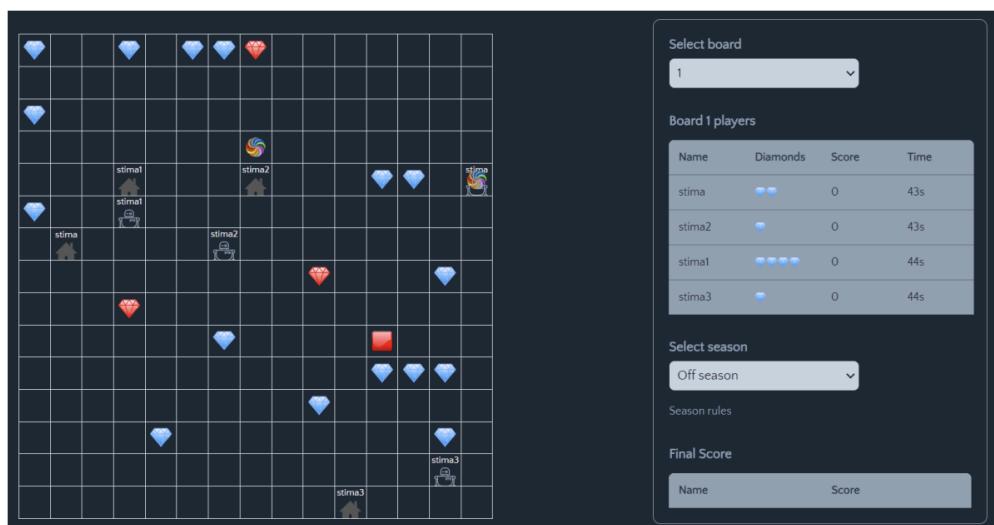
## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>1</b>
<b>BAB I.....</b>	<b>2</b>
<b>BAB II.....</b>	<b>5</b>
2.1 Algoritma Greedy.....	5
2.2 Elemen Algoritma Greedy.....	5
2.3 Game Engine Diamonds.....	5
2.4 Alur Menjalankan Program.....	6
2.4 Alur Pengembangan Logika Bot.....	7
<b>BAB III.....</b>	<b>10</b>
3.1 Alternatif Greedy.....	10
3.1.1 Greedy by Considering Distance - Diamond - Base.....	10
3.1.2 Greedy by Considering Ratio - Distance Base.....	11
3.1.3 Greedy by Considering Diamond Point.....	13
3.1.4 Greedy by Considering Diamond Cluster.....	14
3.2 Strategi Greedy yang Diimplementasikan.....	15
<b>BAB IV.....</b>	<b>16</b>
4.1 Implementasi dalam Pseudocode.....	16
4.2 Struktur Data dalam Program.....	24
4.3 Analisis dan Pengujian.....	26
4.3.1 Inventory Penuh.....	26
4.3.2 Mencari Cluster Terbaik.....	26
4.3.3 Mencari Cluster Terbaik dan Kembali Ke Base.....	27
4.3.4 Red Button.....	28
4.3.5. Teleporter.....	29
4.3.6. Waktu Hampir Habis.....	31
<b>BAB V.....</b>	<b>34</b>
5.1 Kesimpulan.....	34
5.2 Saran.....	34
<b>LAMPIRAN.....</b>	<b>35</b>
Repository.....	35
Youtube.....	35
<b>DAFTAR PUSTAKA.....</b>	<b>36</b>

## 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.



Gambar 1. Ilustrasi Game Diamonds

Komponen-komponen dari permainan Diamonds antara lain:

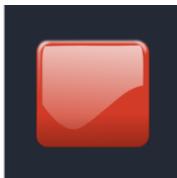
1. Diamonds



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



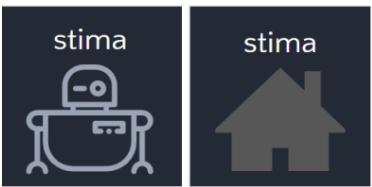
Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

## 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. Bot and Bases



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

## 5. Inventory

Name	Diamonds	Score	Time
stima	♥♥	0	43s
stima2	♥	0	43s
stima1	♥♥♥♥	0	44s
stima3	♥	0	44s

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

Cara kerja permainan diamonds 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 algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) dengan prinsip *take what you can get now*, yaitu mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma ini berusaha untuk mencapai solusi yang optimal dengan melakukan pemilihan yang paling rasional dan efektif pada setiap tahapnya.

#### 2.2 Elemen Algoritma Greedy

Elemen-elemen yang terdapat pada algoritma greedy, antara lain:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap langkah.
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi : menentukan apakah himpunan yang dipilih sudah memberikan solusi
4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu . Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif: fungsi untuk memaksimumkan atau meminimumkan

#### 2.3 Game Engine Diamonds

Untuk memulai menjalankan game dan mengimplementasikan bot, diperlukan starter pack game Galaxio. Starter pack dapat diunduh melalui link 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>

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

Bot starter pack, yang secara umum berisi:

- a. Program untuk memanggil API yang tersedia pada backend
- b. Program bot logic
- c. Program utama (main) dan utilitas lainnya

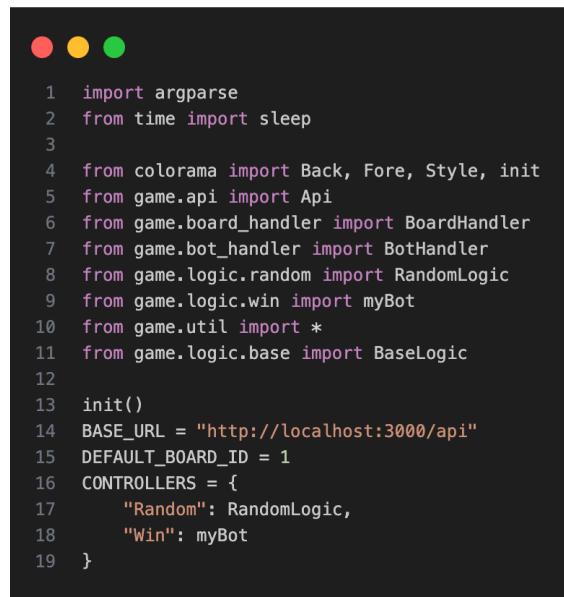
## 2.4 Alur Menjalankan Program

Berikut adalah langkah-langkah untuk menjalankan program:

1. Arahkan directory ke `tubes1-IF2211-game-engine-1.1.0` sebanyak 2 kali lalu ketikkan di terminal `npm run start` dan klik link `localhost` yang muncul di terminal.
2. Setelah `localhost` muncul arahkan directory kembali ke `tubes1-IF2211-bot-starter-pack-1.0.1`
3. Sebelum menjalankan bot etimo diamonds kalian bisa menyesuaikan `script` yang ada pada `run-bots.bat` (Windows) atau `run-bots.sh` (Linux/macOS) dari segi logic yang digunakan, email, nama, dan password.

```
1  #!/bin/bash
2
3  python3 main.py --logic Random --email=coba1@email.com --name=bot1 --password=123456 --team etimo &
4  python3 main.py --logic Win --email=blinkBlink@email.com --name=blinkBlink --password=123456 --team etimo &
5  python3 main.py --logic Random --email=coba3@email.com --name=bot3 --password=123456 --team etimo &
6  python3 main.py --logic Random --email=coba4@email.com --name=bot4 --password=123456 --team etimo &
7  python3 main.py --logic Random --email=coba5@email.com --name=bot5 --password=123456 --team etimo &
```

4. Pastikan juga kalian sudah melakukan import untuk kelas yang telah kalian buat pada `main.py` dan daftarkan pada dictionary CONTROLLERS seperti berikut.

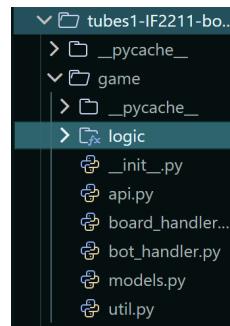


```
1 import argparse
2 from time import sleep
3
4 from colorama import Back, Fore, Style, init
5 from game.api import Api
6 from game.board_handler import BoardHandler
7 from game.bot_handler import BotHandler
8 from game.logic.random import RandomLogic
9 from game.logic.win import myBot
10 from game.util import *
11 from game.logic.base import BaseLogic
12
13 init()
14 BASE_URL = "http://localhost:3000/api"
15 DEFAULT_BOARD_ID = 1
16 CONTROLLERS = {
17     "Random": RandomLogic,
18     "Win": myBot
19 }
```

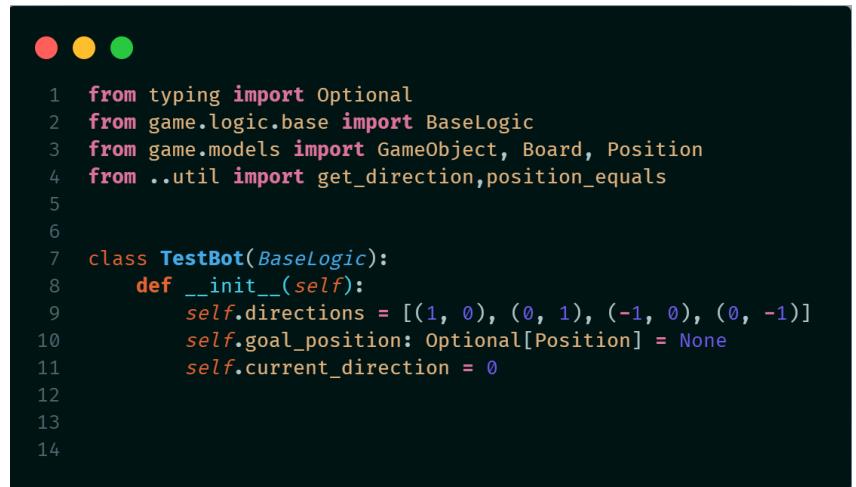
5. Setelah langkah-langkah tersebut dilakukan, bot dapat dimainkan dengan mengetikkan `./run-bots.sh` untuk pengguna Linux/macOS ataupun `./run-bots.bat` untuk pengguna Windows.

## 2.4 Alur Pengembangan Logika Bot

Pada folder “bot-starter-pack” terdapat folder “logic” yang digunakan untuk pengembangan logika bot dalam bahasa pemrograman python.



Pengembangan dilakukan dengan membuat file .py (python) baru didalam folder “logic” dengan cara membuat class Bot baru. Untuk mengakses elemen-elemen yang ada pada permainan, dapat melakukan *import* dari folder “/game/models” atau dari folder “/game/logic”. Pada saat pengembangan logika bot, tidak diperkenankan untuk menambah file baru selain di folder “logic”. Tidak diperkenankan juga untuk menghapus file apapun dari game yang sudah ada.



```

1  from typing import Optional
2  from game.logic.base import BaseLogic
3  from game.models import GameObject, Board, Position
4  from ..util import get_direction, position_equals
5
6
7  class TestBot(BaseLogic):
8      def __init__(self):
9          self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
10         self.goal_position: Optional[Position] = None
11         self.current_direction = 0
12
13
14

```

Berikut merupakan contoh dari pengembangan logika bot baru dengan nama kelas TestBot.



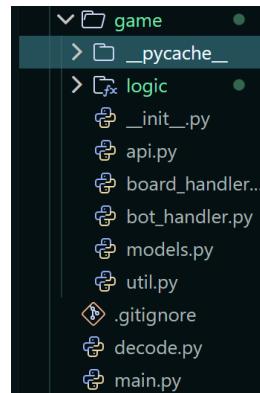
```

1  def next_move(self, board_bot: GameObject, board: Board):
2      props = board_bot.properties
3      # Analyze new state
4      if props.diamonds == 5:
5          # Move to base
6          base = board_bot.properties.base
7          self.goal_position = base
8      else:
9          # Just roam around
10         self.goal_position = None
11
12     current_position = board_bot.position
13     if self.goal_position:
14         # We are aiming for a specific position, calculate delta
15         delta_x, delta_y = get_direction(
16             current_position.x,
17             current_position.y,
18             self.goal_position.x,
19             self.goal_position.y,
20         )
21     else:
22         # Roam around
23         delta = self.directions[self.current_direction]
24         delta_x = delta[0]
25         delta_y = delta[1]
26         if random.random() > 0.6:
27             self.current_direction = (self.current_direction + 1) % len(
28                 self.directions
29             )
30     return delta_x, delta_y

```

Logika utama yang diimplementasikan merupakan fungsi dari “next\_move” yang akan menghasilkan delta\_x dan delta\_y sebagai arah gerak bot. Namun, dalam implementasinya dapat dibuat fungsi-fungsi tambahan sebagai pembantu membuat logika.

Setelah logika bot sudah ada, untuk menjalankan kode tersebut, logika harus ditambahkan pada game engine. Masih pada folder “game” terdapat file “main.py” yang harus ditambahkan.



Berikut merupakan struktur folder dari main.py. Untuk menambahkan logika bot yang dibuat pada game engine, logika harus di *import* ke main.py.

```
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.testBot import TestBot
12 from game.logic.newBot import NewBot
```

Setelah logika bot di*import*, tambahkan nama logika Bot (sesuai dengan nama logika bot yang di*import*) ke bagian controller.

```
1 init()
2 BASE_URL = "http://localhost:3000/api"
3 DEFAULT_BOARD_ID = 1
4 CONTROLLERS = {
5     "Random": RandomLogic,
6     "TestBot" : TestBot,
7     "NewBot" : NewBot,
8 }
```

Nama yang ditambahkan merupakan nama yang akan digunakan saat menjalankan bot. Jika nama sudah ditambahkan, bot sudah bisa dijalankan.

## BAB III

### APLIKASI STRATEGI GREEDY

#### 3.1 Alternatif Greedy

##### 3.1.1 Greedy by Considering Distance - Diamond - Base

Algoritma ini bertujuan untuk mengumpulkan berlian sebanyak mungkin dengan menggunakan pendekatan yang mengutamakan pengambilan keputusan untuk pemilihan tujuan dengan mempertimbangkan jarak dari posisi saat ini ke berlian terdekat dan ke *base*. Selain itu jika inventory hampir penuh akan diutamakan berlian dengan *point 1* dan jika masih sedikit akan bergerak ke blok yang memiliki jumlah berlian terbanyak di sekitarnya.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: Algoritma ini mempertimbangkan berlian mana yang dapat diambil berdasarkan *current condition*, seperti nilai berlian dan jaraknya dari posisi bot.
- b. Himpunan solusi: Langkah-langkah yang diambil oleh bot untuk memilih berlian dan mencapai tujuan tertentu, seperti mengambil berlian dengan *point 1* atau kembali ke *base* untuk menyimpan berlian jika inventori hampir penuh.
- c. Fungsi solusi: Dalam algoritma ini, fungsi solusinya adalah pemilihan berlian terdekat dengan posisi saat ini dan pengambilan keputusan berdasarkan jumlah berlian di inventori.
- d. Fungsi seleksi: Memilih berlian mana yang akan diambil oleh bot berdasarkan kondisi saat ini. Algoritma ini memilih berlian terdekat dengan posisi saat ini atau berlian dengan *point 1* terlebih dahulu, tergantung pada jumlah berlian di inventori.
- e. Fungsi kelayakan: Apakah berlian tersebut dapat diambil oleh bot berdasarkan kondisi saat ini, seperti jaraknya dari posisi bot dan nilai berliannya.

- f. Fungsi obyektif: Mengumpulkan sebanyak mungkin berlian dengan cara yang paling efisien, dengan mempertimbangkan nilai dan jaraknya.

- **Analisis Efisiensi Solusi**

Algoritma ini memiliki kompleksitas waktu  $O(n^2)$ , di mana n adalah jumlah sel di papan permainan. Hal ini disebabkan oleh iterasi melalui setiap sel pada papan permainan dalam pencarian posisi terbaik untuk pergi. Setiap iterasi melakukan pembandingan jarak, yang memakan waktu  $O(n)$  karena algoritma harus membandingkan jarak antara posisi bot dengan setiap sel di papan permainan. Dengan demikian, total waktu yang dibutuhkan untuk melakukan iterasi melalui semua sel menjadi  $O(n^2)$ .

- **Analisis Efektifitas Solusi**

Algoritma ini efektif dalam mengumpulkan berlian karena mengambil pendekatan yang cukup sederhana dan langsung. Strategi ini efektif apabila jumlah berlian di sekitar bot cukup besar dan bot dapat dengan mudah mengambil berlian tanpa terlalu banyak menghabiskan waktu dan kurang efektif jika ada bot lawan yang berusaha untuk men-*tackle* saat ingin balik ke *base* karena tidak di *handle*.

### 3.1.2 Greedy by Considering Ratio - Distance Base

Algoritma ini bertujuan untuk mengumpulkan diamond berdasarkan ratio jarak dibandingkan dengan point diamond. Algoritma ini juga akan mempertimbangkan jarak Diamond dengan ratio tertinggi (point paling banyak dengan jarak terdekat) dengan jarak ke Base. Apabila inventory sudah hampir penuh, maka algoritma akan mengutamakan untuk kembali ke Base apabila jarak ke Base lebih dekat.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: Algoritma ini mempertimbangkan langkah mana yang dapat diambil berdasarkan *current condition*, seperti ratio nilai berlian dengan jaraknya dengan jarak dengan jarak *base*.

- b. Himpunan solusi: Langkah-langkah yang diambil oleh bot untuk memilih berlian dan mencapai tujuan tertentu, seperti mengambil berlian dengan ratio tertinggi atau kembali ke base apabila inventory hampir penuh dan jarak ke base lebih dekat dibandingkan jarak ke berlian.
- c. Fungsi solusi: fungsi solusi pada algoritma ini merupakan mencari diamond dengan rasio tertinggi, ratio tertinggi adalah point terbesar dengan jarak terdekat, algoritma ini juga akan mempertimbangkan jarak base.
- d. Fungsi seleksi: fungsi untuk memilih posisi selanjutnya bot berdasarkan kondisi saat ini. Algoritma ini akan memilih berlian dengan ratio poin/jarak tertinggi maupun kembali ke base saat inventory hampir penuh dan jarak ke base lebih dekat.
- e. Fungsi kelayakan: Mengumpulkan sebanyak mungkin berlian dengan cara yang paling efisien, dengan mempertimbangkan nilai dan jaraknya.

- **Analisis Efisiensi Solusi**

Algoritma ini memiliki kompleksitas waktu  $O(n)$ , di mana n adalah mencari posisi setiap berlian pada papan. Setelah mendapatkan posisi berlain, akan dilakukan perbandingan rasio antar berlian dengan kompleksitas  $O(1)$ . Sehingga secara keseluruhan algoritma ini memiliki kompleksitas  $O(n)$ .

- **Analisis Efektifitas Solusi**

Algoritma ini efektif dalam mengumpulkan berlian karena mempertimbangkan rasio yang paling baik sebagai langkah bot selanjutnya. Strategi ini efektif apabila terdapat banyak berlian di sekitar bot dan base bot memiliki tempat yang strategis (tidak di ujung board). Posisi berlian dan base berpengaruh agar bot bisa langsung mengambil berlian tanpa menghabiskan waktu mobilisasi.

### 3.1.3 Greedy by Considering Diamond Point

Algoritma ini bertujuan untuk mengumpulkan berlian berdasarkan poin berlian.

Algoritma akan mementingkan berlian merah dibandingkan berlian biru karena poin yang lebih besar.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: Algoritma ini mempertimbangkan langkah mana yang dapat diambil berdasarkan *current condition*, seperti posisi berlian-berlian pada board.
- b. Himpunan solusi: Langkah-langkah yang diambil oleh bot untuk memilih berlian dan mencapai tujuan tertentu, seperti mengambil berlian dengan poin tertinggi lalu memilih posisi berlian tersebut sebagai langkah selanjutnya.
- c. Fungsi solusi: fungsi solusi pada algoritma ini merupakan mencari diamond poin tertinggi.
- d. Fungsi seleksi: fungsi untuk memilih posisi selanjutnya bot berdasarkan kondisi saat ini. Algoritma ini akan memilih berlian dengan posisi berlian merah yang merupakan berlian dengan posisi tertinggi.
- e. Fungsi kelayakan: Mengumpulkan sebanyak mungkin berlian dengan cara yang paling efisien, dengan poin dari berlian tersebut.

- **Analisis Efisiensi Solusi**

Algoritma ini memiliki kompleksitas waktu  $O(n)$ , di mana n adalah mencari posisi setiap berlian pada papan. Setelah mendapatkan posisi berlian, akan dilakukan perbandingan poin antar berlian (untuk mencari berlian dengan poin tertinggi) dengan kompleksitas  $O(1)$ . Sehingga secara keseluruhan algoritma ini memiliki kompleksitas  $O(n)$ .

- **Analisis Efektifitas Solusi**

Algoritma ini dinilai kurang efektif karena, rasio munculnya berlian merah (poin lebih tinggi) sangat kecil dibandingkan dengan berlian biasa. Tidak hanya itu, posisi berlian merah tidak selalu merupakan berlian terdekat dari

bot sehingga bot seringkali membuang waktu hanya untuk mobilisasi mengambil berlian merah.

### 3.1.4 Greedy by Considering Diamond Cluster

Algoritma ini bertujuan untuk menentukan langkah selanjutnya dengan mencari daerah dengan berlian paling banyak, lalu mengambil berlian terdekat di daerah tersebut. Jika inventory sudah hampir penuh, algoritma ini akan mencari posisi berlian terdekat dari base.

- **Mapping Elemen Greedy**

- a. Himpunan kandidat: Algoritma ini mempertimbangkan langkah mana yang dapat diambil berdasarkan *current condition*, seperti daerah mana yang memiliki berlian paling banyak, dan berlian terdekat mana yang ada pada daerah tersebut.
- b. Himpunan solusi: Langkah-langkah yang diambil oleh bot untuk memilih berlian dan mencapai tujuan tertentu, seperti mengambil daerah dengan jumlah berlian terbanyak dan mempertimbangkan berlian mana yang terdekat pada daerah tersebut.
- c. Fungsi solusi: fungsi solusi pada algoritma ini merupakan mencari daerah dengan berlian terbanyak.
- d. Fungsi seleksi: fungsi untuk memilih posisi selanjutnya bot berdasarkan kondisi saat ini. Algoritma ini akan memilih daerah dengan berlian paling banyak dan selanjutnya memilih berlian dengan jarak terdekat.
- e. Fungsi kelayakan: Mengumpulkan sebanyak mungkin berlian dengan cara yang paling efisien, dengan poin dari berlian tersebut.

- **Analisis Efisiensi Solusi**

Algoritma ini memiliki kompleksitas waktu  $O(n)$ , di mana n adalah mencari daerah terbaik. Pada daerah terbaik akan dilakukan perbandingan antar jarak berlian dengan kompleksitas  $O(1)$ . Sehingga secara keseluruhan algoritma ini memiliki kompleksitas  $O(n)$ .

- **Analisis Efektifitas Solusi**

Algoritma ini efektif karena mempertimbangkan daerah mana memiliki berlian paling banyak dan setelah mendapatkan daerah tersebut algoritma ini mempertimbangkan berlian terbaik mana yang harus dipilih. Algoritma ini akan efektif jika ada banyak berlian pada suatu daerah tertentu. Algoritma ini juga akan efektif jika jarak base dan juga posisi bot dekat dengan daerah berlian yang dipilih.

### 3.2 Strategi Greedy yang Diimplementasikan

Berdasarkan alternatif-alternatif greedy pada 3.1, dapat dilihat bahwa tiap alternatif memiliki efektivitasnya masing-masing. Dari alternatif-alternatif tersebut, strategi yang diimplementasikan adalah “Greedy by Considering Diamond Cluster” yang dipadukan dengan “Greedy by Considering Ratio - Distance Base”. Beberapa poin penting dari implementasi bot antara lain:

1. Jika inventory sudah penuh bot akan kembali ke pada base
2. Jika inventory sudah hampir penuh, bot akan mencari daerah mana yang terdekat dari base dan memiliki diamond yang paling banyak
3. Saat berada di daerah terbaik, bot akan mencari berlian dengan rasio tertinggi sebagai posisi selanjutnya (agar berlian dapat diambil)
4. Jika tidak ada daerah berlian yang optimal di sekitar area bot, dan terdapat “Tombol Merah” yang dekat, maka bot akan menekan tombol tersebut.
5. Setiap mobilisasi yang dilakukan oleh bot akan mempertimbangkan jarak menggunakan “Teleporter” ataupun tidak, jika lebih dekat menggunakan “Teleporter”, maka bot akan pindah ke posisi teleporter.
6. Jika waktu permainan sudah hampir habis maka, bot akan kembali ke base (tidak memedulikan berapa banyak berlian yang ada pada inventory)

## BAB IV

### IMPLEMENTASI DAN UJI COBA

#### 4.1 Implementasi dalam *Pseudocode*

Menentukan arah gerak selanjutnya

```
function get_direction(current_x: integer, current_y: integer,
dest_x: integer, dest_y: integer, h_priority: boolean) ->
(delta_x, delta_y)
{ Menentukan arah gerak selanjutnya berdasarkan prioritas arah
sekarang. Fungsi clamp sudah terdefinisi. }

Deklarasi

Algoritma
    delta_x <- clamp(dest_x - current_x, -1, 1)
    delta_y <- clamp(dest_y - current_y, -1, 1)
    if h_priority then
        if delta_x != 0 then
            delta_y <- 0
        endif
    else
        if delta_y != 0 then
            delta_x <- 0
        endif
    endif
    return (delta_x, delta_y)
```

Menentukan jarak langsung dari satu titik ke titik lain

```
function distance(p1: Position, p2: Position) -> integer
{ Fungsi nilai mutlak (abs) sudah terdefinisi. }

Deklarasi

Algoritma
    return abs(p1.x - p2.x) + abs(p1.y - p2.y)
```

Menentukan jarak langsung dari asal ke tujuan melalui *teleporter*

```
function distance_tp(src: Position, dest: Position, close_tp:
Position, far_tp: Position) -> integer

Deklarasi

Algoritma
    return distance(src, close_tp) + distance(far_tp, dest)
```

Menentukan jarak terdekat yang perlu dilalui dari asal ke tujuan

```

function min_distance(src: Position, dest: Position, close_tp:
Position, far_tp: Position) -> integer
{ Fungsi nilai minimum (min) sudah terdefinisi. }

Deklarasi

Algoritma
    return min(distance(src, dest), distance_tp(src, dest,
close_tp, far_tp))

```

## Menentukan posisi-posisi teleporter

```

function teleporter_positions(board: Board) -> array of Position

Deklarasi
    teleporters : array of Position
    game_object : GameObject

Algoritma
    teleporters <- {}
    for game_object in board.game_objects do
        if (game_object.type = "TeleportGameObject") then
            teleporters <- teleporters U (game_object.position)
        endif
    endfor
    return teleporters

```

## Menentukan apakah posisi bot terhalang oleh teleporter

```

function blocked_by_teleporter(position: Position, board: Board)
-> boolean
{ Fungsi position_equals sudah terdefinisi. Asumsi selalu hanya
ada sepasang teleporter. }

Deklarasi
    teleporters : array of Position

Algoritma
    teleporters <- teleporter_positions(board)
    return (position_equals(teleporters[0], position) or
    position_equals(teleporters[1], position))

```

## Menentukan posisi red button

```

function red_button(board_bot: GameObject, board: Board,
close_tp: Position, far_tp: Position) -> Position
{ Mengembalikan posisi red button. Jika red button lebih dekat
ketika melewati teleporter dan posisi sekarang bukan di
teleporter, fungsi mengembalikan posisi teleporter terdekat. }

Deklarasi
    game_object : GameObject

```

```

red_button : Position
total_distance_normal, total_distance_tp : integer

Algoritma
  for game_object in board.game_objects do
    if (game_object.type = "DiamondButtonGameObject") then
      red_button <- game_object.position
      break
    endif
  endfor

  total_distance_normal <- distance(board_bot.position,
red_button)
  total_distance_tp <- distance(board_bot.position, close_tp)
+ distance(far_tp, red_button)

  if ((total_distance_normal <= total_distance_tp) or
position_equals(board_bot.position, close_tp)) then
    return red_button
  else
    return close_tp
  endif

```

Menentukan apakah suatu titik berada dalam *cluster* titik lain

```

function within_cluster(p1: Position, p2: Position) -> boolean
{ fungsi nilai mutlak (abs) sudah terdefinisi. Daerah cluster
suatu titik adalah dua satuan secara horizontal dan dua satuan
secara vertikal. }

Deklarasi

Algoritma
  return (abs(p1.x - p2.x) < 3 and abs(p1.y - p2.y) < 3)

```

Menentukan jumlah poin yang bisa didapatkan dalam suatu *cluster*

```

function count_diamond_cluster(diamond: GameObject, diamonds:
array of GameObject) -> integer

Deklarasi
  sum : integer
  diamond2 : GameObject

Algoritma
  sum <- diamond.properties.points
  for diamond2 in diamonds do
    if (within_cluster(diamond.position, diamond2.position)
and not(position_equals(diamond.position,
diamond2.position))) then
      sum <- sum + diamond2.properties.points
    endif
  endfor
  return sum

```

Menentukan *cluster* terbaik

```

function best_cluster(board_bot: GameObject, board: Board,
close_tp: Position, far_tp: Position) -> Position
{ Menentukan posisi cluster tersebut lebih
dekat tercapai jika melewati teleporter dan posisi sekarang bukan
di teleporter, fungsi mengembalikan posisi teleporter terdekat. }

Deklarasi
    diamonds : array of GameObject
    props : Properties
    best_cluster, diamond : GameObject
    min_distance_points_ratio, total_distance_normal,
    total_distance_tp, total_distance : integer
    through_tp : boolean

Algoritma
    diamonds <- board.diamonds
    props <- board_bot.properties

    best_cluster <- diamonds[0]
    min_distance_points_ratio <- 10000
    through_tp <- False

    for diamond in diamonds do:
        if (props.diamonds == 4 and diamond.properties.points ==
        2) then { diamond tidak bisa diambil }
            continue
        endif
        total_diamonds <- count_diamond_cluster(diamond,
        diamonds)

        total_distance_normal <- distance(board_bot.position,
        diamond.position)
        total_distance_tp <- distance(board_bot.position,
        close_tp) + distance(far_tp, diamond.position)
        total_distance <- min(total_distance_normal,
        total_distance_tp)

        { Penyebut kondisional di bawah merupakan poin aktual
        yang bisa didapatkan dari cluster. }
        if ((total_distance / min(total_diamonds,
        props.inventory_size - props.diamonds)) <
        min_distance_points_ratio) then
            min_distance_points_ratio <- total_distance /
            min(total_diamonds, props.inventory_size -
            props.diamonds)
            best_cluster <- diamond
            if total_distance = total_distance_tp then
                through_tp <- true
            else
                through_tp <- false
            endif
        endif
    endfor

```

```

{ Kondisi kedua dibuat agar bot tidak diarahkan ke
  posisinya sendiri. }
if (through_tp and
not(position_equals(board_bot.position, close_tp))) then
  return close_tp
else
  return best_cluster.position
endif

```

Menentukan *cluster* terbaik dengan mempertimbangkan jarak ke *base*

```

function best_cluster(board_bot: GameObject, board: Board,
close_tp: Position, far_tp: Position) -> Position
{ Menentukan posisi cluster terbaik dengan mempertimbangkan jarak
ke base. Jika cluster tersebut lebih dekat tercapai jika melewati
teleporter dan posisi sekarang bukan di teleporter, fungsi
mengembalikan posisi teleporter terdekat. }

Deklarasi
  diamonds : array of GameObject
  props : Properties
  best_cluster, diamond : GameObject
  min_distance_points_ratio, total_distance_normal,
  total_distance_tp, total_distance : integer
  through_tp : boolean

Algoritma
  diamonds <-board.diamonds
  props <- board_bot.properties

  best_cluster <- diamonds[0]
  min_distance_points_ratio <- 10000
  through_tp <- False

  for diamond in diamonds do:
    if (props.diamonds == 4 and diamond.properties.points ==
    2) then { diamond tidak bisa diambil }
      continue
    endif
    total_diamonds <- count_diamond_cluster(diamond,
    diamonds)

    d_current_diamond <-min_distance(board_bot.position,
    diamond.position, close_tp, far_tp)
    d_diamond_base <- min_distance(diamond.position,
    board_bot.properties.base, close_tp, far_tp)
    total_distance <- d_current_diamond + d_diamond_base

    { Penyebut kondisional di bawah merupakan poin aktual
      yang bisa didapatkan dari cluster. }
    if ((total_distance / min(total_diamonds,
    props.inventory_size - props.diamonds)) <
    min_distance_points_ratio) then
      min_distance_points_ratio <- total_distance /
      min(total_diamonds, props.inventory_size -

```

```

        props.diamonds))
best_cluster <- diamond
if total_distance = total_distance_tp then
    through_tp <- true
else
    through_tp <- false
endif
endif
endfor

{ Kondisi kedua dibuat agar bot tidak diarahkan ke
posisinya sendiri. }
if (through_tp and
not(position_equals(board_bot.position, close_tp))) then
    return close_tp
else
    return best_cluster.position
endif

```

Mengarahkan *bot* ke *base*

```

function to_base(board_bot: GameObject, board: Board, close_tp:
Position, far_tp: Position) -> Position

Deklarasi
    base : Position
    return_distance_normal, return_distance_tp : integer

Algoritma
    base <- board_bot.properties.base

    return_distance_normal <- distance(board_bot.position, base)
    return_distance_tp <- distance(board_bot.position, close_tp)
    + distance(far_tp, base)

    if (return_distance_normal < return_distance_tp
or position_equals(board_bot.position, close_tp)) then
        return base
    else
        return close_tp
    endif

```

Menentukan gerakan selanjutnya yang terbaik (fungsi utama)

```

function next_move(self, board_bot: GameObject, board: Board) ->
(delta_x, delta_y)
{ Menentukan gerakan selanjutnya yang terbaik. }

Deklarasi
    props : Properties
    current_position, base, close_tp, far_tp, new_position :
Position
    distance_to_base, d_red_button, d_current_cluster,
d_cluster_base, d_red_button_base, d_best_cluster :

```

```

integer

Algoritma
{ Pengambilan informasi berbagai state dari permainan. }
props <- board_bot.properties
current_position <- board_bot.position
base <- board_bot.properties.base

teleporters <- teleporter_positions(board)
if (distance(board_bot.position, teleporters[0]) <
    distance(board_bot.position, teleporters[1])) then
    close_tp <- teleporters[0]
    far_tp <- teleporters[1]
else
    close_tp <- teleporters[1]
    far_tp <- teleporters[0]
endif

distance_to_base <- min(distance(current_position, base),
    distance_tp(current_position, base, close_tp, far_tp))
d_red_button <- distance(red_button(board_bot, board,
    close_tp, far_tp), current_position)

if (props.diamonds = 5) then { Go to base. }
    self.goal_position <- to_base(board_bot, board,
        close_tp, far_tp)
else if (props.diamonds >= 3) then { Find best cluster while
    returning }
    cluster <- best_cluster_base(board_bot, board,
        close_tp, far_tp)
    red <- red_button(board_bot, board, close_tp,
        far_tp)

    d_current_cluster <- min_distance(current_position,
        cluster, close_tp, far_tp)
    d_cluster_base <- min_distance(cluster, base,
        close_tp, far_tp)
    d_red_button_base <- min_distance(red, base,
        close_tp, far_tp)

    if (d_red_button < d_current_cluster and (d_red_button +
        d_red_button_base) < (distance_to_base + 5) and
        not(position_equals(red, close_tp))) then
        self.goal_position = red # If red button is nearest
    else if (d_current_cluster + d_cluster_base) <
        distance_to_base + 5 and not(position_equals(cluster,
            close_tp)) then
        self.goal_position <- cluster { If taking a detour
            to the cluster doesn't take too long }
    else
        { If everything else is too far }
        self.goal_position = to_base(board_bot, board,
            close_tp, far_tp)
    endif
else
    { Search for best cluster or nearest red button. }
    d_best_cluster <- distance(best_cluster(board_bot,
        board, close_tp, far_tp), current_position)

```

```

    if (d_red_button < d_best_cluster and
        not(position_equals(red_button(board_bot, board,
                                         close_tp, far_tp), close_tp))) then
        self.goal_position <- red_button(board_bot, board,
                                         close_tp, far_tp)
    else
        self.goal_position <- best_cluster(board_bot, board,
                                             close_tp, far_tp)
    endif
endif

if ((props.milliseconds_left / 1000) - 2 <
distance_to_base) then { Return to base in time }
    self.goal_position <- to_base(board_bot, board,
                                   close_tp, far_tp)
endif

if self.goal_position then
    { Aiming for a specific position, calculate delta }
    delta_x, delta_y <- get_direction(
        current_position.x,
        current_position.y,
        self.goal_position.x,
        self.goal_position.y,
        self.h_priority
    )
    new_position <- Position(x=(current_position.x +
delta_x), y=(current_position.y + delta_y))
    { Check if destination is not a teleporter, but
blocked by a teleporter }
    if (blocked_by_teleporter(new_position, board) and
not(position_equals(self.goal_position, close_tp))) then
        if (delta_x != 0) then
            { If initially going horizontally, move vertically }
            delta_x <- 0
            self.h_priority <- true
            if (board.is_valid_move(current_position,
delta_x, 1)) then
                delta_y <- 1
            else
                delta_y <- -1
            endif { Go down if unable to go up (at the top
of the board) }
        else
            delta_y <- 0
            self.h_priority <- false
            if (board.is_valid_move(current_position, 1,
delta_y)) then
                delta_x <- 1
            else
                delta_x <- -1 { Go left if unable to go
right (at the right edge of the board) }
            endif
        endif
    endif
else
    { Roam around }

```

```

    delta = self.directions[self.current_direction]
    delta_x = delta[0]
    delta_y = delta[1]
    if random.random() > 0.6 then
        self.current_direction = (self.current_direction +
        1) % len(self.directions)
    endif
endif
return delta_x, delta_y

```

## 4.2 Struktur Data dalam Program

Implementasi logika bot dilakukan dengan bahasa pemrograman Python. Struktur data program didefinisikan pada *file models.py* yang merepresentasikan struktur data dalam bentuk *class*. Kelas-kelas yang terdapat pada *file* ini adalah:

- Class Bot
 

Kelas ini merupakan kelas untuk objek-objek *bot* dalam permainan. Informasi yang disimpan dalam kelas ini berupa identitas (atribut *name* dan *email*) serta *id* bot tersebut.
- Class Position
 

Kelas ini merepresentasikan suatu koordinat/posisi dengan absis ditandai dengan atribut *x* dan ordinat ditandai dengan atribut *y*.
- Class Base
 

Kelas ini merupakan kelas turunan dari Position. Kelas ini merepresentasikan posisi/koordinat *base* suatu bot.
- Class GameObject
 

Kelas ini merupakan kelas untuk objek-objek yang muncul pada permainan. Atribut yang dimiliki oleh kelas ini meliputi *id*, *position* (menandakan posisi objek tersebut), *type*, dan *properties*. Nilai atribut *type* menentukan tipe objek tersebut. Nilai atribut *type* yang mungkin meliputi BotGameObject, DiamondGameObject, TeleportGameObject, dan DiamondButtonGameObject.

Atribut *properties* berisi sifat-sifat yang dimiliki oleh GameObject tersebut.
- Class Properties
 

Kelas ini berisi informasi tentang sifat yang dimiliki suatu GameObject. Atribut yang dimiliki kelas ini meliputi *points*, *pair\_id*, *diamonds*, *score*, *name*, *inventory\_size*, *can\_tackle*, *milliseconds\_left*, *time\_joined*, dan *base*.

Atribut-atribut tersebut tidak harus ada semua dalam suatu GameObject.

Atribut Properties yang dimiliki oleh GameObject bergantung pada *type* GameObject tersebut.

Atribut *points* menandakan jumlah poin yang bisa didapatkan dari GameObject tersebut. Atribut *pair\_id* dimiliki oleh *teleporter* untuk menentukan *teleporter* yang berpasangan. Atribut *diamonds* adalah jumlah diamonds yang dimiliki oleh suatu *bot*. Atribut *score* adalah skor yang dimiliki suatu *bot*. Atribut *name* merupakan nama dari GameObject tersebut. Atribut *inventory\_size* menandakan ukuran inventory dari *bot*. Atribut *can\_tackle* menentukan apakah suatu *bot* dapat melakukan *tackle* kepada *bot* lain. Atribut *milliseconds\_left* adalah sisa waktu sebelum *bot* keluar dari permainan. Atribut *time\_joined* menunjukkan waktu bergabung suatu GameObject. Atribut *base* menyimpan posisi *base* suatu *bot*.

- Class Config

Kelas ini berisi informasi tentang konfigurasi yang dipakai untuk permainan. Konfigurasi tersebut mengatur hal-hal seperti ukuran *inventory* dan rasio *diamond* merah. Atribut kelas ini meliputi *generation\_ratio*, *min\_ratio\_for\_generation*, *red\_ratio*, *seconds*, *pairs*, *inventory\_size*, dan *can\_tackle*.

- Class Feature

Kelas ini memiliki dua atribut, yaitu nama (*name*) dan konfigurasi (*config*).

- Class Board

Kelas ini berisi informasi tentang papan yang dipakai pada permainan. Atribut yang dimiliki kelas ini meliputi *id*, *height*, *weight*, *features*, *minimum\_delay\_between\_moves*, dan *game\_objects*. Selain itu, kelas ini juga memiliki beberapa metode seperti *diamonds()* (mengembalikan daftar *diamond* pada papan), *get\_bot()*, (mengembalikan daftar *bot* pada papan), dan *is\_valid\_move()* (menentukan apakah suatu gerakan *valid* dan menampilkan pesan kesalahan jika tidak *valid*).

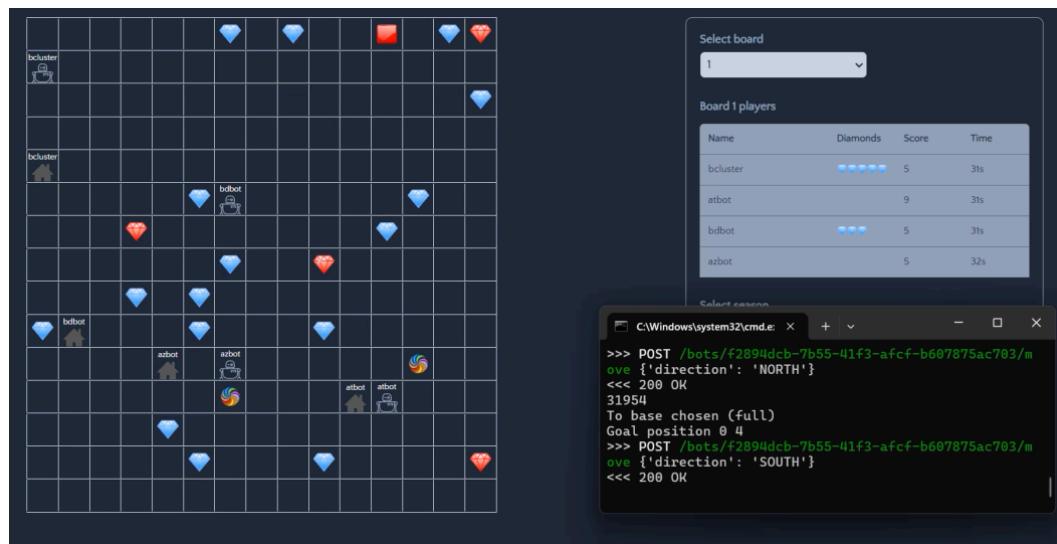
### 4.3 Analisis dan Pengujian

Analisis dan pengujian terhadap *bot* yang dipilih dilakukan dengan mempertarungkan *bot* (yang bernama *bcluster*) dalam analisis dengan variasi bot

lainnya yang dibuat oleh kelompok kami. Pada pengujian ini, *bcluster* melawan tiga bot lainnya, yaitu *bot bdbot*, *bot azbot*, dan *bot atbot*. Aksi-aksi yang dilakukan oleh *bcluster* akan diuji. Untuk mempermudah analisis, informasi tambahan terkait aksi yang dipilih *bot* akan ditampilkan melalui *terminal*.

#### 4.3.1 Inventory Penuh

Bagian ini akan menguji aksi yang dipilih oleh *bot* ketika sudah memiliki *inventory* penuh.

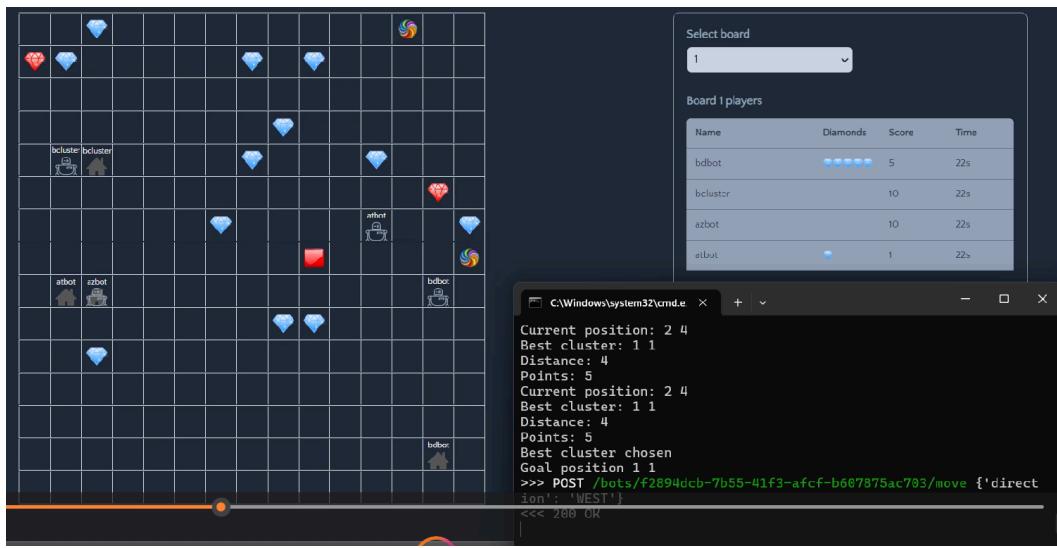


Gambar 4.3.1.1. Kondisi bot ketika *inventory* sudah penuh

Dapat dilihat bahwa bot akan memilih untuk pergi ke *base* ketika *inventory* sudah penuh. Pada kasus seperti ini, strategi *greedy* yang diberikan sudah optimal.

#### 4.3.2 Mencari Cluster Terbaik

Bagian ini akan menguji kasus ketika diamond yang dimiliki *bot* kurang dari tiga. Bot akan mencari *cluster* yang memiliki perbandingan jarak dan jumlah poin yang bisa didapatkan dari *cluster* terkecil.

Gambar 4.3.2.1. Kondisi bot ketika mencari *cluster* terbaik

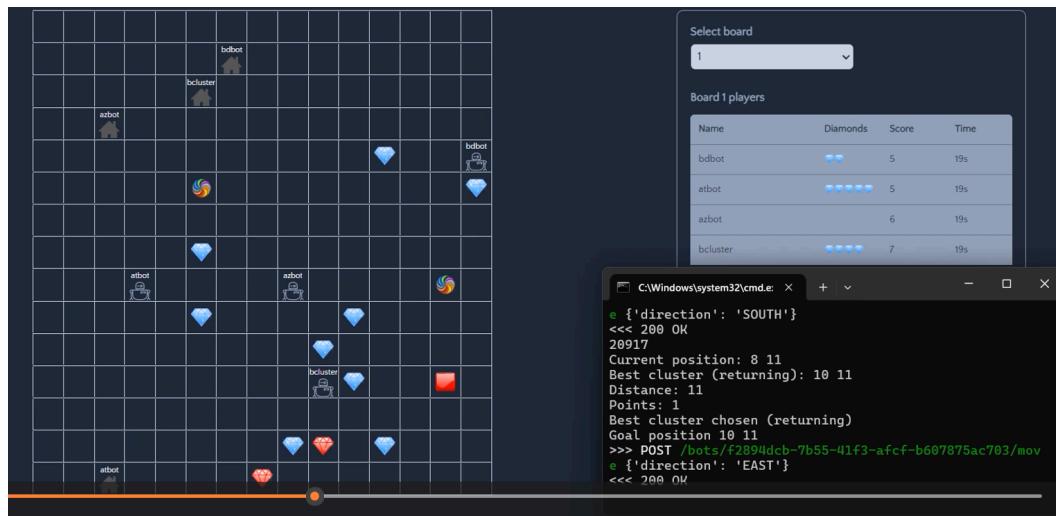
Pada gambar 4.3.2.1., terlihat bahwa bot memilih posisi (1, 1) sebagai tujuannya. Hal tersebut dilakukan karena *cluster* pada posisi (1, 1) memiliki nilai *min\_distance\_points\_ratio* sebesar  $(4 / 5) = 0.8$ . Pada kasus seperti ini, terlihat bahwa strategi *greedy* yang dihasilkan berhasil memberi hasil yang optimal.

Kasus yang menyebabkan strategi ini tidak optimal adalah saat *red button* dilangkahi oleh *bot* lain dan mengacak posisi *diamond*. Kasus lain adalah ketika *bot* lain sudah mencapai daerah tujuan *bcluster* sehingga *diamond* pada daerah tersebut sudah dimakan *bot* lain. Hal-hal tersebut akan memaksa *bot* untuk mencari ulang *cluster* lain yang terbaik. Pada kasus seperti ini, strategi yang dihasilkan belum optimal.

### 4.3.3 Mencari Cluster Terbaik dan Kembali Ke Base

Bagian ini akan menguji kasus ketika isi *diamond* pada *inventory* bot berisi tiga atau lebih. Mekanisme yang diharapkan adalah *bot* akan kembali ke *base* dengan mengambil *detour* untuk mengisi sisa *inventory* yang kosong ketika jarak *detour* yang diambil tidak terlalu besar.

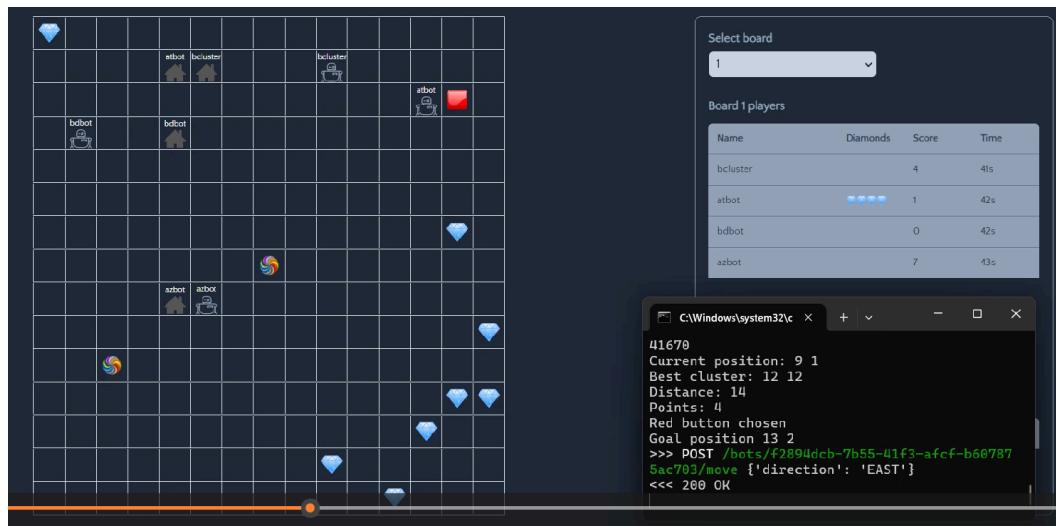
Pada gambar 4.3.3.1., terlihat bahwa *bot* memilih untuk pergi ke titik (10, 11) terlebih dahulu karena jarak *detour* yang diambil untuk mengambil diamond di titik (10, 11) tidak melebihi batas. Dalam kasus ini, batas memiliki nilai sebesar <jarak ke *base* + 5>. Pada kasus tersebut, strategi yang diberikan oleh *bot* telah optimal.

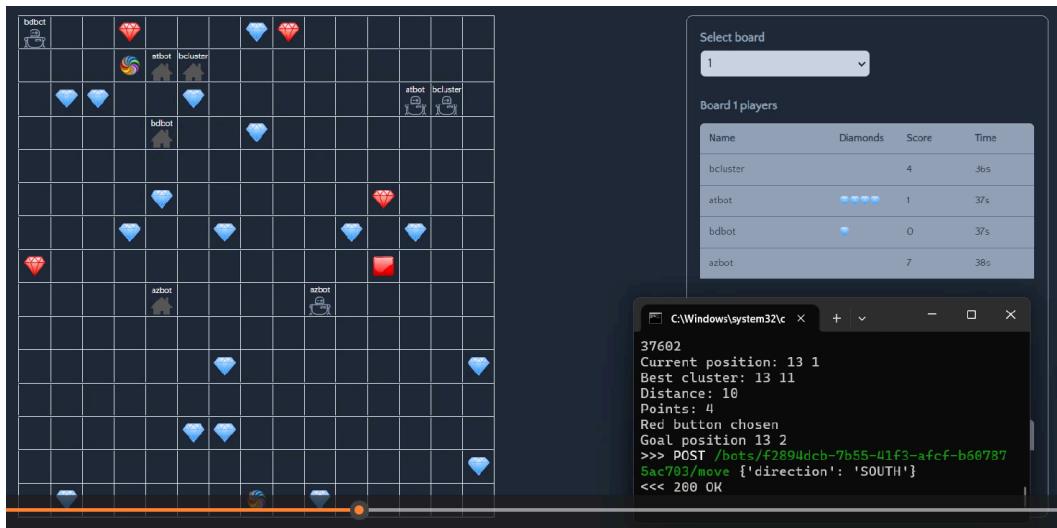
Gambar 4.3.3.1. Kondisi *bot* ketika kembali ke *base* dengan mengambil *diamond* sepanjang jalan

Kasus yang menyebabkan strategi ini tidak optimal sama seperti kasus pada mencari *cluster* terbaik, yaitu saat *red button* dilangkahi oleh *bot* lain dan mengacak posisi *diamond* serta ketika *bot* lain sudah mencapai daerah tujuan *bcluster* sehingga *diamond* pada daerah tersebut sudah dimakan *bot* lain. Hal-hal tersebut akan memaksa *bot* untuk mencari ulang *cluster* lain yang terbaik. Pada kasus seperti ini, strategi yang dihasilkan belum optimal.

#### 4.3.4 Red Button

Bagian ini akan menguji kasus ketika *bot* menargetkan *red button*. Bot akan menargetkan *red button* jika *red button* lebih jauh daripada *cluster* terbaik.

Gambar 4.3.4.1. Kondisi *bot* ketika memilih *red button*

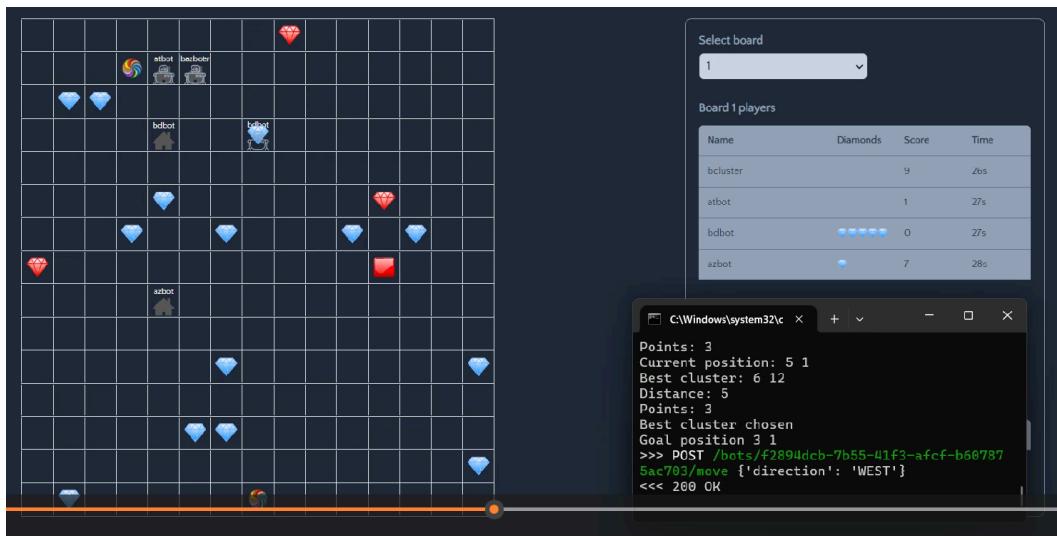
Gambar 4.3.4.2. Kondisi *bot* setelah memilih *red button*.

Pada gambar 4.3.4.1, informasi tambahan mengenai pilihan dan alur berpikir *bot* terdapat pada terminal. Terlihat bahwa posisi *cluster* terbaik berada di koordinat (12, 12) dan berjarak 14 satuan dari bot, sedangkan posisi *red button* hanya berjarak 5 satuan. Oleh karena itu, *bot* memilih untuk pergi ke *red button*. Terlihat juga bahwa *diamond* yang berada di sekitar *base* sangat sedikit.

Pada gambar 4.3.4.2, terdapat kondisi *board* setelah *bot* menginjak *red button*. Terlihat bahwa jumlah *diamond* yang terletak di dekat *base* lebih banyak sehingga untuk selanjutnya, *bot* tidak perlu pergi terlalu jauh dari *base* untuk mendapatkan *diamond*. Dapat disimpulkan bahwa implementasi logika *red button* sudah optimal.

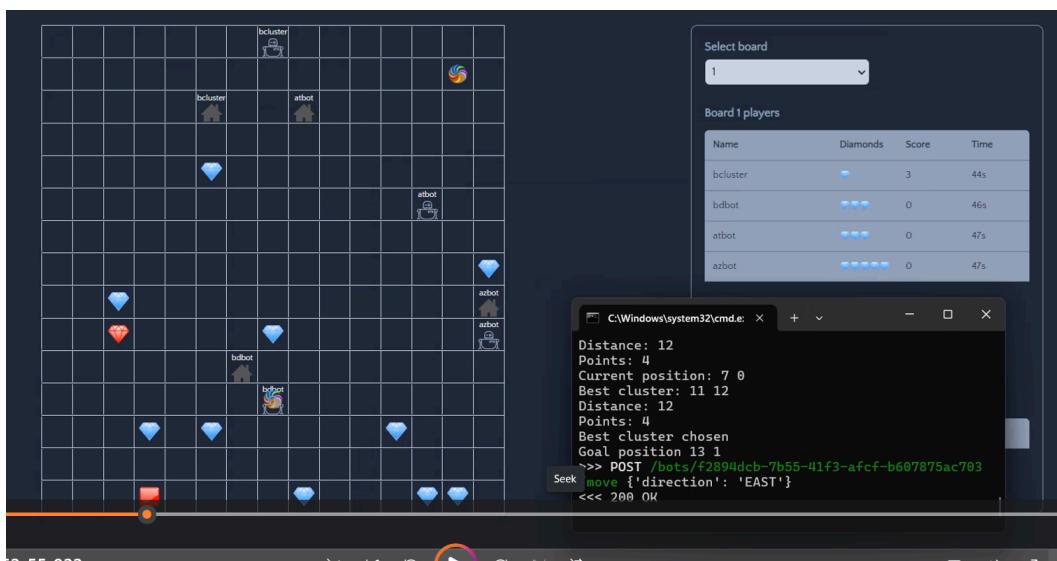
### 4.3.5. Teleporter

Bagian ini akan menguji apakah *bot* sudah memanfaatkan *teleporter* dengan baik. Indikator pemakaian *teleporter* dengan baik adalah saat *bot* memanfaatkan *teleporter* jika suatu koordinat tujuan lebih cepat dicapai dengan melewati *teleporter*.

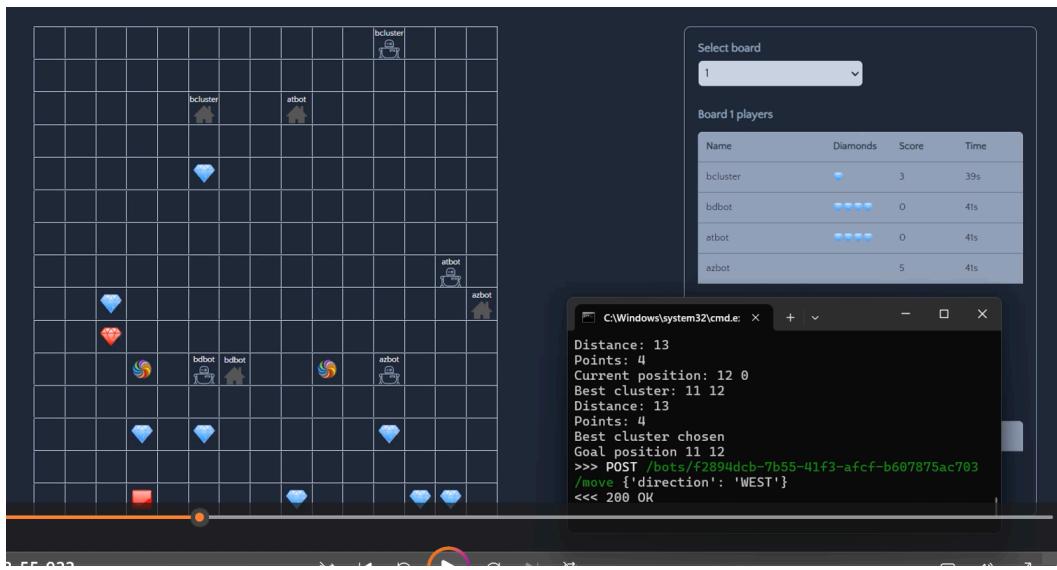


Gambar 4.3.5.1. Pemanfaatan *teleporter* untuk mendapatkan *cluster* terbaik.

Pada gambar tersebut, sekilas terlihat bahwa *cluster* terdekat yang bisa dipilih *bot* adalah pada titik (2, 2) yang memiliki nilai *min\_distance\_points\_ratio* sebesar  $(4 / 2) = 2$ . Akan tetapi, jika dilihat lebih lanjut, terdapat *cluster* pada posisi (6, 12) yang memiliki nilai *min\_distance\_points\_ratio* sebesar  $(5 / 3) = 1.67$  dengan *distance* sebesar 5 satuan jika melewati *teleporter*. Nilai *cluster* tersebut dinilai lebih baik oleh *bot* sehingga algoritma akan mengarahkan *bot* ke posisi *teleporter* terdekat. Pada kasus ini, strategi *greedy* berhasil mendapat nilai optimum.



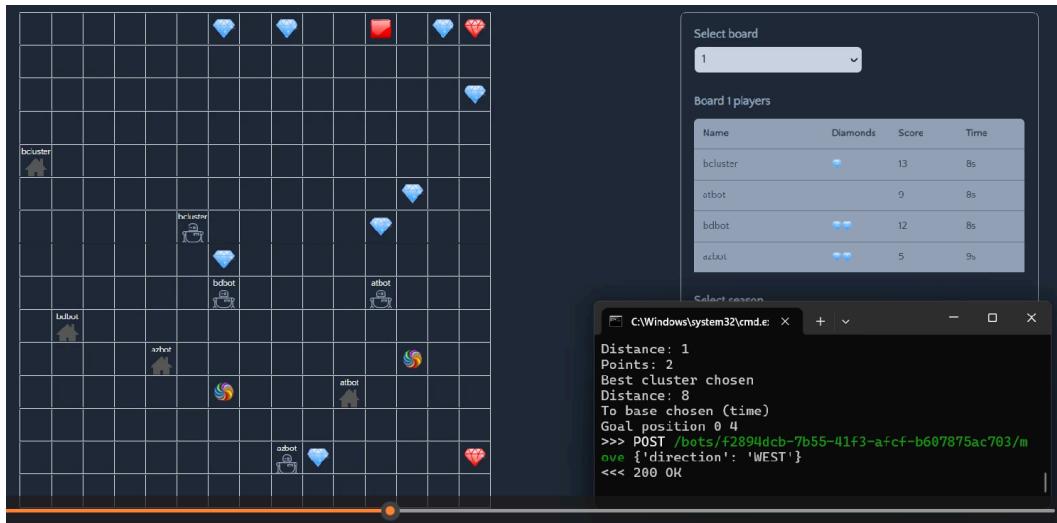
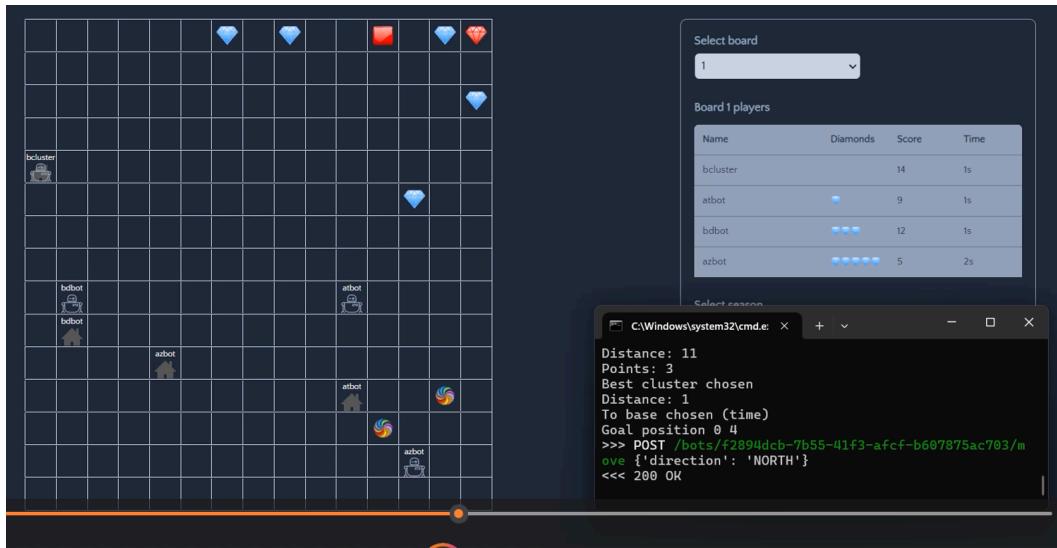
Gambar 4.3.5.2. Tujuan awal *bcluster* berupa *teleporter* pada posisi (13, 1)

Gambar 4.3.5.3. Posisi tujuan *bcluster* berubah karena *teleporter* yang berpindah

Suatu masalah yang timbul dalam memanfaatkan *teleporter* adalah ketika posisi *teleporter* berpindah sebelum digunakan. Hal tersebut berpotensi menyia-nyiakan langkah yang sudah diambil *bot* dan menyebabkan *bot* perlu mencari ulang solusi yang terbaik. Pada kasus ini, strategi yang diimplementasikan belum optimal. Masalah ini dapat memengaruhi efektivitas strategi yang lain (mencari *cluster* terbaik, kembali ke *base*, dll.) sehingga tidak menjadi optimal. Contoh kasus tersebut dapat dilihat di gambar 4.3.5.2. dan 4.3.5.3.

#### 4.3.6. Waktu Hampir Habis

Bagian ini akan menguji perilaku *bot* ketika waktu hampir habis. Mekanisme kembali ke *base* yang dipakai oleh *bot* ini adalah kembali ketika  $\langle \text{sisa waktu (detik)} \rangle + 2 \leq \langle \text{jarak ke base} \rangle$ .

Gambar 4.3.6.1. Kondisi *bot* ketika waktu hampir habisGambar 4.3.6.2. Kondisi *bot* yang berhasil sampai ke *base* dengan tepat waktu

Pada gambar 4.3.6.1., terlihat bahwa waktu *bot* tersisa 8 detik dan jarak dari *bot* ke *base* sebesar 7 satuan. Terminal menunjukkan bahwa *bot* akan pergi ke *base*. Gambar 4.3.6.2. menunjukkan bahwa *bot* berhasil sampai ke *base* tepat waktu. Pada kasus ini, strategi *greedy* yang diimplementasikan sudah optimal.

Ada beberapa kasus yang menyebabkan strategi ini kurang optimal. Kasus pertama adalah ketika *bot* ingin memanfaatkan *teleporter* untuk sampai ke *base* dengan lebih cepat, namun *teleporter* tersebut berpindah tempat sebelum *bot* sampai. Kasus ini dapat menyebabkan gagalnya *bot* untuk sampai ke *base* dengan tepat waktu. Kasus kedua adalah ketika suatu *teleporter* menghalangi jalan

kembali *bot*. Terhalangnya *bot* oleh *teleporter* akan memaksa *bot* untuk menghindari *teleporter* dengan bergeser secara horizontal atau vertikal (tergantung arah awal *bot*). Hal tersebut menyebabkan adanya maksimum dua langkah tambahan yang tidak terhitung oleh bot (langkah pertama untuk menghindari *teleporter* dan langkah kedua untuk kembali ke *path* awal). Pada kedua kasus tersebut, mekanisme kembali ke *base* menjadi tidak optimal.

## BAB V

### KESIMPULAN, SARAN

#### 5.1 Kesimpulan

Dalam permainan diamonds, setiap bot ditempatkan secara acak dengan home base, score, dan inventory awal nol. Bot memiliki waktu bergerak untuk mengumpulkan diamond berwarna merah (2 poin) dan biru (1 poin) dengan tujuan memaksimalkan skor. Inventory bot berfungsi sebagai penyimpanan sementara, yang perlu dikosongkan di home base. Interaksi antar bot terjadi jika satu bot menimpa yang lain, menyebabkan bot yang terkena (*tackle*) ke home base dengan kehilangan semua diamond pada inventonya. Permainan diimplementasikan algoritma greedy pada permainan ini sebagai logika dari bot.

Algoritma greedy merupakan algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*). Pada permainan ini, implementasi greedy yang dipakai merupakan “Greedy by Considering Diamond Cluster” dan “Greedy by Considering Diamond Ration - Distance Base.” Bot mengambil keputusan seperti kembali ke base ketika inventory penuh, mencari daerah terdekat dengan banyak diamond saat inventory hampir penuh, dan memilih berlian dengan rasio tertinggi. Bot juga memanfaatkan fitur tambahan seperti "Tombol Merah" dan "Teleporter" untuk keuntungan strategis. Pada akhirnya, jika waktu hampir habis, bot fokus kembali ke base tanpa memperdulikan jumlah berlian di inventory.

#### 5.2 Saran

Terdapat beberapa saran yang dapat dilakukan untuk pengembangan lebih lanjut, antara lain:

1. Mempertimbangkan cara untuk *tackle* bot lawan dalam logika permainan.
2. Membuat kode yang lebih efisien dengan struktur kode yang lebih *readable*.

## **LAMPIRAN**

### **Repository**

Link Repository dari Tugas Besar 01 IF2211 Strategi Algoritma kelompok 15 “blinkBlink” adalah sebagai berikut.

[https://github.com/angiekriera/Tubes1\\_blinkBlink.git](https://github.com/angiekriera/Tubes1_blinkBlink.git)

### **Youtube**

Link video Youtube dari Tugas Besar 01 IF2211 Strategi Algoritma kelompok 15 “blinkBlink” adalah sebagai berikut.

<https://youtu.be/rUnQVG90bWg>

## DAFTAR PUSTAKA

[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), terakhir diakses 7 Maret 2024.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf), terakhir diakses 8 Maret 2024.

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf), terakhir diakses 9 Maret 2024.