

PEMANFAATAN ALGORITMA *GREEDY* DALAM APLIKASI PERMAINAN “DIAMONDS”

Diajukan sebagai pemenuhan tugas besar I.



Oleh:

Kelompok 03 (**metiuganteng**)

13522029 - Ignatius Jhon Hezkiel Chan

13522043 - Daniel Mulia Putra Manurung

13522093 - Matthew Vladimir Hutabarat

Dosen Pengampu : Dr. Ir. Rinaldi, M.T.

IF2211 - Strategi Algoritma

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

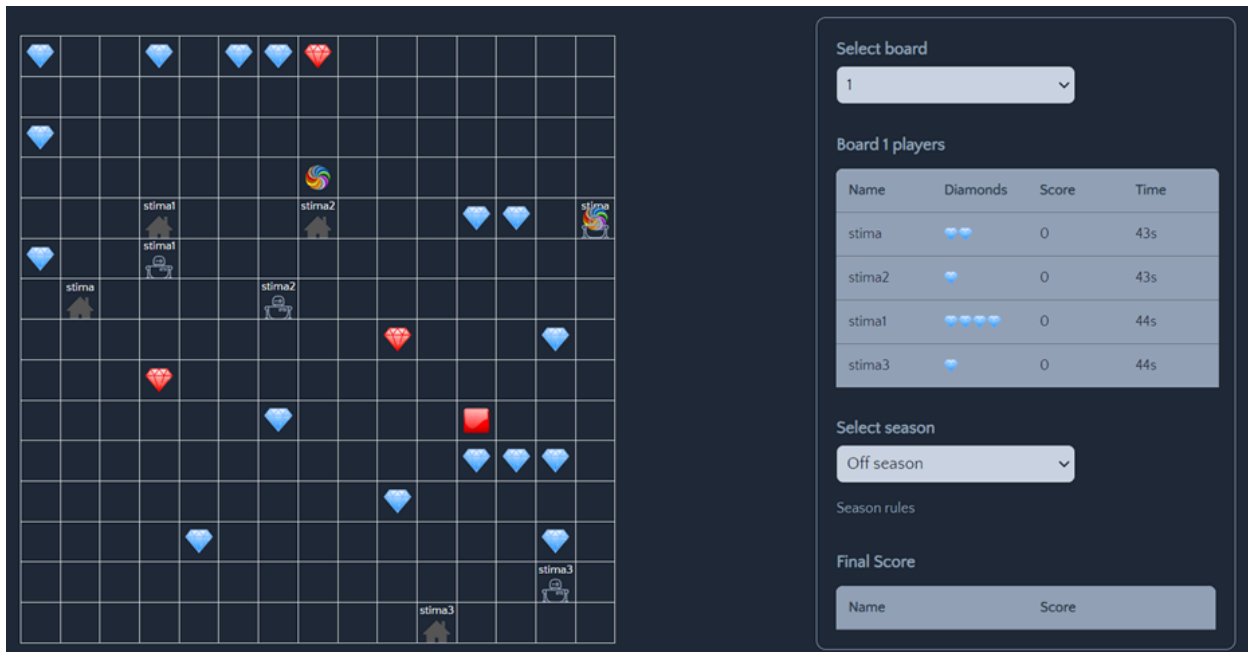
DAFTAR ISI

DAFTAR ISI	2
BAB 1 DESKRIPSI TUGAS	3
BAB 2 TEORI DASAR	7
2.1. Algoritma Greedy	7
2.2. Eksekusi bot pada Diamonds	8
BAB 3 APLIKASI STRATEGI GREEDY	10
3.1. Mapping Persoalan Diamonds ke Elemen Algoritma Greedy	10
3.2. Eksplorasi Alternatif Solusi Greedy	14
3.3. Analisis Efisiensi dan Efektivitas Alternatif Solusi	19
3.4. Pemilihan Strategi Greedy	22
BAB 4 IMPLEMENTASI DAN PENGUJIAN	23
4.1. Implementasi Algoritma Greedy pada Bot Diamonds	23
4.2. Struktur Data	31
4.3. Pengujian dan Analisis Desain	36
BAB 5 PENUTUP	38
5.1. Kesimpulan	38
5.2. Saran	38
5.3. Komentar dan Refleksi	39
LAMPIRAN	40
DAFTAR PUSTAKA	41

BAB 1

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 sederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.



The screenshot displays the Diamonds game interface. On the left is a 20x20 grid representing the game board. It contains several blue diamonds, red diamonds, and obstacles (represented by house icons). Three player bots are visible, labeled 'stima', 'stima2', and 'stima3'. On the right is a sidebar with the following sections:

- Select board:** A dropdown menu showing '1'.
- Board 1 players:** A table showing the performance of the four bots.
- Select season:** A dropdown menu showing 'Off season'.
- Season rules:** A section for game rules.
- Final Score:** A table for the final scores.

Name	Diamonds	Score	Time
stima	2	0	43s
stima2	1	0	43s
stima1	4	0	44s
stima3	1	0	44s

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

Program permainan Diamonds terdiri atas:

1. Game engine, yang secara umum berisi:
 - a. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot
 - b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan

2. Bot starter pack, yang secara umum berisi:
 - a. Program untuk memanggil API yang tersedia pada backend
 - b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)
 - c. Program utama (main) dan utilitas lainnya

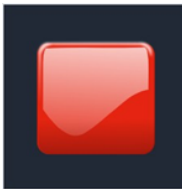
Komponen-komponen dari permainan Diamond antara lain:

1. Diamonds



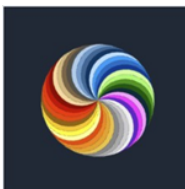
Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio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

2. Red Button / Diamond Button



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

3. Teleporters



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

4. Bots and Bases



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

5. Inventory

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

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

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

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.

4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

BAB 2

TEORI DASAR

2.1. Algoritma Greedy

Algoritma greedy adalah algoritma yang paling populer untuk menemukan solusi optimasi dengan cepat. Persoalan optimasi terbagi menjadi 2 macam persoalan, yaitu memaksimalkan dan meminimalkan sesuatu dengan cara yang efisien. Algoritma ini memecahkan persoalan secara langkah per langkah sedemikian sehingga, pada setiap langkah, algoritma akan mengambil pilihan terbaik yang diperoleh pada saat tersebut tanpa memperhatikan konsekuensi ke depan (prinsip “*take what you can get now!*”) dan “berharap” untuk langkah kedepannya mendapat langkah optimum lokal hingga berakhir dengan optimum global.

Elemen - elemen pada algoritma greedy :

1. Himpunan kandidat (C) : berisi kandidat yang akan dipilih pada setiap langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb).
2. Himpunan solusi (S) : berisikan kandidat yang sudah dipilih.
3. Fungsi solusi : menentukan apakah himpunan kandidat 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 (layak atau tidak)
6. Fungsi obyektif: memaksimumkan atau meminimumkan

Beberapa contoh persoalan populer yang dapat diselesaikan dengan greedy sebagai berikut:

1. Huffman Coding

Digunakan untuk kompresi data. Ini adalah metode yang efisien untuk mengurangi ukuran data dengan meng-assign kode biner yang lebih pendek untuk karakter yang lebih sering muncul di teks.

2. Kruskal Algorithm

Biasa digunakan untuk persoalan pohon minimal (Minimum Spanning Tree) pada grafik. Algoritma ini membantu menemukan subset dari semua edge dalam grafik yang membentuk pohon tanpa siklus dengan total bobot yang minimal.

3. Prim's Algorithm

Seperti Kruskal, Prim's Algorithm juga digunakan dalam masalah pohon minimal, tetapi fokus pada membangun pohon dari satu simpul awal dengan memilih edge terkecil yang terhubung.

4. Dijkstra's Algorithm

Algoritma ini digunakan untuk mencari jalur terpendek antara simpul awal dalam graf dengan simpul lainnya.

2.2. Eksekusi bot pada Diamonds

Pada permainan diamonds, permainan dilakukan berbasis website, sehingga setiap aksi yang dilakukan mulai dari mendaftarkan bot hingga menjalankan aksi bot akan memerlukan HTTP request terhadap API endpoint tertentu yang disediakan backend.

Program akan mengkalkulasikan move selanjutnya secara berkala berdasarkan nilai yang diberikan oleh fungsi `next_move` dari kelas `KielBot`. Setelah mendapat nilai dari `next_move`. Program akan mengirimkan POST request terhadap endpoint `/api/bots/{id}/move` dengan body berisi direction yang akan ditempuh selanjutnya ("NORTH", "SOUTH", "EAST", atau "WEST"). Apabila berhasil, maka backend akan memberikan response code 200 dengan body berisi kondisi board setelah move tersebut. Langkah ini akan dilakukan terus menerus hingga waktu bot habis. Jika waktu bot habis, bot secara otomatis akan dikeluarkan dari board. Program juga dapat mendapatkan informasi terbaru pada board dengan mengirimkan GET request terhadap endpoint `/api/boards/{id}` sehingga tampilan board pada frontend akan selalu ter-update.

Program akan ditulis dalam bahasa Python. Untuk menjalankan bot dapat mengikuti langkah berikut :

1. Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-bot-starter-pack-1.0.1
```

2. Install dependencies menggunakan pip

```
pip install -r requirements.txt
```

3. Untuk menjalankan suatu bot harus menggunakan email dan nama yang berbeda untuk setiap botnya

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

Untuk menjalankan beberapa bot sekaligus (pada contoh ini, kita menjalankan 4 bot dengan logic yang sama, yaitu `game/logic/random.py`)

- a. Untuk windows


```
./run-bots.bat
```

- b. Untuk Linux / (possibly) macOS

```
./run-bots.sh
```

Selanjutnya, implementasi bot dengan algoritma greedy dapat dilakukan pada permainan Diamonds dengan langkah - langkah berikut.

1. Buat file baru pada direktori/game/logic (misalnya mybot.py)
2. Buat kelas yang meng-inherit kelas BaseLogic, lalu implementasikan constructor dan method next_move pada kelas tersebut

```
mybot.py U X
game > logic > mybot.py > ...
1  from game.logic.base import BaseLogic
2  from game.models import Board, GameObject
3
4
5  class MyBot(BaseLogic):
6      def __init__(self):
7          # Initialize attributes necessary
8          self.my_attribute = 0
9
10     def next_move(self, board_bot: GameObject, board: Board):
11         # Calculate next move
12         delta_x = 1
13         delta_y = 0
14         return delta_x, delta_y
15
```

3. Import kelas yang telah dibuat pada main.py dan daftarkan pada dictionary

```
main.py M X
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.mybot import MyBot
12
13 init()
14 BASE_URL = "http://localhost:3000/api"
15 DEFAULT_BOARD_ID = 1
16 CONTROLLERS = {"Random": RandomLogic, "MyBot": MyBot}
```

BAB 3

APLIKASI STRATEGI *GREEDY*

3.1. Mapping Persoalan Diamonds ke Elemen Algoritma Greedy

Himpunan Kandidat	Seluruh diamond yang terdapat pada arena permainan Diamonds.
Himpunan Solusi	Objek yang menjadi target.
Fungsi Solusi	Mengarahkan bot dan memberikan aksi apa yang akan dilakukan kepada target
Fungsi Seleksi (<i>selection function</i>)	Melakukan seleksi dari seluruh objek yang terdapat dalam arena permainan Diamonds yang berhubungan dengan algoritma <i>greedy</i> yang digunakan.
Fungsi Kelayakan (<i>feasible</i>)	Melakukan validasi kelayakan terhadap objek - objek dalam arena permainan Diamonds, hasil dari penerapan fungsi kelayakan akan digunakan dalam fungsi seleksi.
Fungsi Obyektif	Mengumpulkan sebanyak mungkin poin sebelum timer selesai.

3.2. Eksplorasi Alternatif Solusi Greedy

a. Alternatif 1

1. Penjelasan Umum Strategi

Salah satu alternatif algoritma yang kami gunakan adalah *greedy by three step ahead value*. Jenis greedy ini adalah akan menemukan langkah terbaik untuk bot bergerak dengan memperhatikan 3 diamond dengan value tertinggi. Value didapatkan dengan membagi total besar points dengan jarak tempuh ke objek objek yang diinginkan. Algoritma ini akan

mencari 3 diamond dengan value terbesar dengan pemain. Kemudian, ketiga diamond tersebut akan dicari lagi 3 diamond dengan value terbesar dan seterusnya. Algoritma akan dilakukan secara rekursif hingga didapatkan 3 diamond (ada 3^3 kemungkinan). Setiap kemungkinan akan dicatat kedalam list GameObject dan list tuple yang berisikan point dan movenya lalu akan dimasukkan ke dalam list besar yang menyimpan data tersebut. Setelah selesai dari rekursif, list besar tersebut akan diolah dan dijumlahkan point dan move lalu akan dibagi total point dengan total move. Bot akan menjadikan rute 3 diamond dengan value terbesar menjadi tujuannya.

2. Mapping Elemen Algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan Diamonds.
Fungsi Kelayakan (<i>feasible</i>)	Mencari objek yang memiliki tipe 'DiamondGameObject' sehingga punya atribut nilai poin
Fungsi Seleksi (<i>selection function</i>)	Mengurutkan objek objek berdasarkan rasio poin/jarak mulai dari terbesar hingga terkecil lalu memilih 3 objek dengan rasio terbesar dan melakukan hal yang sama untuk 3 objek tersebut hingga didapatkan 3 langkah diamond.
Fungsi Solusi	Memilih arah pergerakan bot sesuai dengan hasil objek target yang didapatkan, dan mengarahkan bot pemain ke lokasi tersebut.
Himpunan Solusi	Objek yang menjadi target.
Fungsi Obyektif	Mengumpulkan sebanyak mungkin poin sebelum timer selesai.

3. Implementasi Algoritma

```

def next_move(self, board_bot: GameObject, board: Board):
# Himpunan Kandidat
    props = board.game_objects
    currentPosition = board_bot.position
    base = board_bot.properties.base
# Himpunan kelayakan
    Ndiamond = board_bot.properties.diamonds
    if (Ndiamond== 5 or Ndiamond == 4):
        self.goal_position = base
        delta_x, delta_y = get_direction(
            currentPosition.x,
            currentPosition.y,
            self.goal_position.x,
            self.goal_position.y,
        )
    else :
        copyBoard: list[GameObject] = []
        for diamond in board.diamonds:
            copyBoard.append(diamond)

        bestmove = []
        AllBestMove : list[list[GameObject]] = []

# Fungsi seleksi
briliant_move(copyBoard,currentPosition,bestmove,AllBestMove)

        MaxBestMove = [0,1]
        elmt = 0

        for i in range(len(AllBestMove)):
            PointMove = [0,0]
            for j in range(len(AllBestMove[i])):
                PointMove[0] +=
AllBestMove[i][j].properties.points
                if (j == 0):
                    PointMove[1] +=

```

```

move2Diamond(currentPosition, AllBestMove[i][j].position)
        else:
            PointMove[1] +=
move2Diamond(AllBestMove[i][j-1].position, AllBestMove[i][j].position)

            if (MaxBestMove[0]/MaxBestMove[1] >
PointMove[0]/PointMove[1]):
                MaxBestMove[0] = PointMove[0]
                MaxBestMove[1] = PointMove[1]
                elmt = i
            elif(MaxBestMove[0]/MaxBestMove[1] ==
PointMove[0]/PointMove[1] and MaxBestMove[0] < PointMove[0]):
                MaxBestMove[0] = PointMove[0]
                MaxBestMove[1] = PointMove[1]
                elmt = i
# Fungsi solusi
        self.goal_position = AllBestMove[elmt][0].position

        delta_x, delta_y = get_direction(
            currentPosition.x,
            currentPosition.y,
            self.goal_position.x,
            self.goal_position.y,
        )
        return delta_x, delta_y

```

b. Alternatif 2

1. Penjelasan Umum Strategi

Salah satu alternatif algoritma yang dapat digunakan adalah *greedy by value*, *greedy by distance*. Greedy by value adalah pendekatan algoritma greedy yang memprioritaskan value dari objek dalam arena Diamonds yang dapat menjadi kandidat target. Algoritma greedy akan melakukan seleksi dari seluruh objek yang terdapat dalam arena, dan memprioritaskan objek yang memiliki poin tertinggi. Greedy by distance adalah pendekatan algoritma greedy yang memprioritaskan distance dari objek dalam arena Diamonds yang dapat menjadi kandidat target. Distance ini juga mempertimbangkan adanya teleporter yang dapat

menghasilkan jarak yang lebih dekat. Algoritma greedy akan melakukan seleksi dari seluruh objek yang terdapat dalam arena, dan mencari objek dengan jarak terdekat dengan bot user. Dengan kedua pendekatan greedy ini, akan dicari rasio value/distance yang terbaik.

2. Mapping Elemen Algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan Diamonds.
Fungsi Kelayakan (<i>feasible</i>)	Mencari objek yang memiliki tipe diamonds sehingga memiliki poin yang dapat didapatkan.
Fungsi Seleksi (<i>selection function</i>)	Mengurutkan objek - objek berdasarkan rasio jarak/poin mulai dari yang terkecil hingga terbesar, lalu memilih objek dengan rasio terkecil.
Fungsi Solusi	Memilih arah pergerakan bot sesuai dengan hasil objek target yang didapatkan, dan mengarahkan bot pemain ke lokasi tersebut.
Himpunan Solusi	Objek yang menjadi target
Fungsi Obyektif	Mengumpulkan sebanyak mungkin poin sebelum timer selesai.

3. Implementasi Algoritma

```
def next_move(self, board_bot: GameObject, board: Board):
    # Himpunan Kandidat
    props = board_bot.properties

    # Fungsi kelayakan
    if props.diamonds == 5 or props.diamonds == 4:
```

```

        base = board_bot.properties.base
        self.goal_position = base
    else:
        self.goal_position = None

    current_position = board_bot.position
    if self.goal_position:
        delta_x, delta_y = get_direction(
            current_position.x,
            current_position.y,
            self.goal_position.x,
            self.goal_position.y,
        )
    else:
        diamond_list = board.diamonds
        objective = []
        tele = [d for d in board.game_objects if d.type ==
"TeleportGameObject"]

# Fungsi seleksi
        for jewel in diamond_list:
            att = (jewel.position.x, jewel.position.y,
jewel.properties.points/(((jewel.position.x - current_position.x)**2 +
(jewel.position.y - current_position.y)**2)**(1/2)))
            objective.append(att)
            if (tele[0].position.x != current_position.x and
tele[0].position.y != current_position.y and tele[1].position.x !=
current_position.x and tele[1].position.y != current_position.y):
                att = (tele[0].position.x, tele[0].position.y,
jewel.properties.points/((((tele[0].position.x -
current_position.x)**2 + (tele[0].position.y -
current_position.y)**2)**(1/2)))+((((tele[1].position.x -
jewel.position.x)**2 + (tele[1].position.y -
jewel.position.y)**2)**(1/2))))
                objective = sorted(objective, key = lambda x:x[2],
reverse=True)
            print(objective)

# Fungsi solusi
        delta_x = get_direction(current_position.x,

```

```

current_position.y, objective[0][0], objective[0][1])[0]
        delta_y = get_direction(current_position.x,
current_position.y, objective[0][0], objective[0][1])[1]

        return delta_x, delta_y

```

c. Alternatif 3

1. Penjelasan Umum Strategi

Salah satu alternatif algoritma yang dapat digunakan adalah *greedy by value*, *greedy by distance*, *greedy by time* dan *greedy by tackle*. Greedy by value dan greedy by distance pada alternatif ini mirip dengan alternatif 2, tetapi dengan beberapa tambahan. Pada alternatif 3, greedy by value dan distance akan memperhitungkan diamond lain yang berjarak cukup dekat dengan titik bot saat ini, dan akan mengambil diamond tersebut. Bot juga akan mengunjungi base, jika base terletak di antara titik bot saat ini dan diamond tujuan, dikarenakan dengan demikian bot akan melewati jalur dengan jarak yang sama, tetapi dapat menyimpan diamond di base. Greedy by time adalah pendekatan greedy yang mempertimbangkan waktu sisa permainan, pada algoritma ini bot akan mengubah cara pencariannya dengan mengambil diamond dengan value/jarak terbesar, dan langsung kembali ke base, atau diamond lain yang ada di sekitarnya saja. Greedy by tackle adalah salah satu pendekatan algoritma greedy yang mengutamakan mengambil poin bot pemain lain. Algoritma ini akan mencari bot pemain lain yang terdapat dalam arena, dan akan mencoba untuk melakukan tackle dan mengambil poin yang telah dikumpulkan oleh bot dari pemain lain tersebut, jika bot tersebut terletak di antara bot pemain dan posisi tujuan, dengan demikian bot pemain akan melewati jalur berbeda dengan total jarak yang sama.

2. Mapping Elemen Algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan Diamonds.
Fungsi Kelayakan (<i>feasible</i>)	Mencari objek selain objek bot pemain sendiri, dan memisahkan objek sesuai tipenya dalam beberapa array yang berbeda.
Fungsi Seleksi (<i>selection function</i>)	Mengurutkan objek - objek berdasarkan rasio jarak/poin mulai dari yang terkecil hingga terbesar, lalu memilih objek dengan rasio terkecil.
Fungsi Solusi	Memilih arah pergerakan bot sesuai dengan hasil objek target yang didapatkan, dan mengarahkan bot pemain ke lokasi tersebut.
Himpunan Solusi	Objek yang menjadi target
Fungsi Obyektif	Mengumpulkan sebanyak mungkin poin sebelum timer selesai.
Tambahan: Strategi Time	Bot akan memilih diamond dengan jarak tersingkat dari titik bot sekarang, dan diamond di sekitarnya, dan langsung menuju ke base.
Tambahan: Strategi Tackle	Jika dalam salah satu rute kemungkinan dari titik bot saat ini ke tujuan terdapat bot player lain, maka bot akan mengambil path yang mana bot player lain tersebut dapat dijangkau, dan akan mencoba untuk melakukan tackle dan mengambil diamond yang telah dikumpulkan bot tersebut.

3. Implementasi Program

```
def next_move(self, board_bot: GameObject, board: Board):

    # Himpunan Kandidat
    teleports = []
    bots = []
    diamonds = []
    shuffle = 0
    totalDist = 0

    # Fungsi Kelayakan
    for x in board.game_objects:
        if(x.type == "BotGameObject"):
            bots.append(x)
        elif x.type == "DiamondGameObject":
            diamonds.append(x)
        elif x.type == "TeleportGameObject":
            teleports.append(x.position)
        elif x.type == "DiamondButtonGameObject":
            shuffle = x.position

    props = board_bot.properties
    botPos = board_bot.position
    base = props.base
    toBase = bestRoute(teleports,base,botPos)

    # Fungsi Seleksi
    if props.diamonds==5:
        self.goal = toBase
    elif props.diamonds<3 and (props.milliseconds_left//1000>=20):
        diamonds = sorted(diamonds,key= lambda x:
bestRatio(x.properties.points,x.position,botPos,diamonds,teleports))
        self.goal =
bestGoal(teleports,diamonds,bots,botPos,props.diamonds)
    else:
        diamonds = sorted(diamonds,key= lambda x:
```

```

(bestRoute(teleports,x.position,botPos)[0]+bestRoute(teleports,x.position,base)[0],bestRoute(teleports,x.position,botPos)[0]))
    temp =
bestGoal(teleports,diamonds,bots,botPos,props.diamonds)
    self.goal = temp
    if props.diamonds>=3:
        if not betweenPoint(botPos,toBase[1],temp[1]) and
extraMove(toBase[1],botPos,temp[1])>2:
            self.goal = toBase

        if self.goal[0]>=bestRoute(teleports,shuffle,botPos)[0]+5 and
props.diamonds<3:
            self.goal = bestRoute(teleports,shuffle,botPos)

        if betweenPoint(self.goal[1],botPos,toBase[1]) and
not(botPos.x==base.x and botPos.y==base.y):
            self.goal = toBase

    FromDestToBase =
self.goal[0]+bestRoute(teleports,base,self.goal[2])[0]
    if FromDestToBase>(props.milliseconds_left//1000) and
props.diamonds>0 and not betweenPoint(toBase[1],botPos,self.goal[1]):
        self.goal = toBase

# Fungsi Solusi
    delta_x,delta_y = get_direction(
        botPos.x,
        botPos.y,
        self.goal[1].x,
        self.goal[1].y,
        teleports,
        bots,
    )
    return delta_x, delta_y

```

3.3. Analisis Efisiensi dan Efektivitas Alternatif Solusi

a. Alternatif 1

1. Kompleksitas Algoritma

Kompleksitas terbesar dalam algoritma alternatif satu berada pada proses sorting dan rekursi objek yang terdapat dalam arena. Algoritma sorting ini mempunyai kompleksitas $O(N^2)$ dan dalam rekursi mempunyai kompleksitas $O(N^2)$. Sehingga bersama sama kompleksitas total dari algoritma greedy alternatif satu adalah $O(N^2)$.

2. Pembuktian berdasarkan Eksperimen Singkat

Sebagai bot stima3 dan melawan 2 ReferenceBot	 <p>Peringkat 3</p>
Sebagai bot stima3 dan melawan 2 ReferenceBot	 <p>Peringkat 3</p>

3. Hasil Analisis

Algoritma ini tidak selalu menghasilkan solusi optimal. Terdapat banyak kekurangan dari algoritma ini, salah satunya jika diamond didalam board kurang dari 3 maka bot terpaksa mencari algoritma lain seperti jarak terdekat dengan bot atau tidak algoritma akan error dan menyebabkan invalid move.

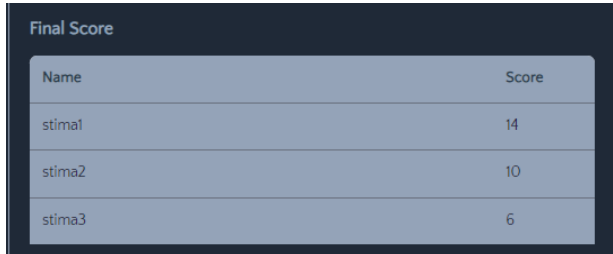
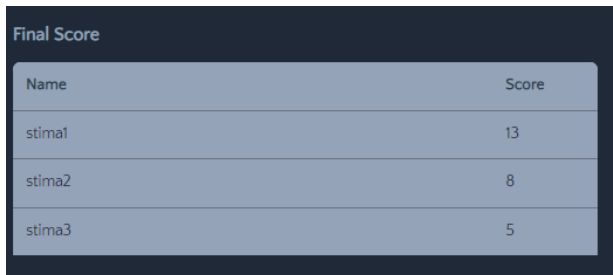
b. Alternatif 2

1. Kompleksitas Algoritma

Kompleksitas terbesar dalam alternatif dua terdapat pada pencarian objek dan pada pengisian kandidat. Penerapan sorting memiliki kompleksitas

$O(N^2)$, dan pada pengisian kandidat memiliki kompleksitas $O(N)$. Sehingga dengan penggabungan seluruh algoritma memiliki kompleksitas $O(N^2)$.

2. Pembuktian berdasarkan Eksperimen Singkat

Sebagai bot stima2 dan melawan 2 ReferenceBot	 <p>Peringkat 2</p>
Sebagai bot stima2 dan melawan 2 ReferenceBot	 <p>Peringkat 2</p>

3. Hasil Analisis

Algoritma ini lebih baik dibandingkan algoritma greedy alternatif satu dikarenakan menghasilkan jumlah diamond yang lebih banyak dan lebih konsisten dibandingkan algoritma greedy sebelumnya. Beberapa kekurangan algoritma ini adalah masih kurangnya optimalisasi waktu yang menyebabkan terkadang terdapat diamond dalam inventory yang tidak sempat diantar ke base, dan tidak adanya awareness mengenai bot pemain lain dalam arena.



c. Alternatif 3

1. Kompleksitas Algoritma

Kompleksitas terbesar dalam alternatif tiga terdapat pada sorting. Penerapan sorting memiliki kompleksitas $O(N^2)$ dan hanya dilakukan satu

kali saja untuk setiap pencariannya, sehingga algoritma alternatif tiga ini memiliki kompleksitas $O(N^2)$.

2. Pembuktian berdasarkan Eksperimen Singkat

Sebagai bot stima1 dan melawan 2 ReferenceBot	 <p>Final Score</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Score</th> </tr> </thead> <tbody> <tr> <td>stima1</td> <td>14</td> </tr> <tr> <td>stima2</td> <td>10</td> </tr> <tr> <td>stima3</td> <td>6</td> </tr> </tbody> </table> <p>Peringkat 1</p>	Name	Score	stima1	14	stima2	10	stima3	6
Name	Score								
stima1	14								
stima2	10								
stima3	6								
Sebagai bot stima1 dan melawan 2 ReferenceBot	 <p>Final Score</p> <table border="1"> <thead> <tr> <th>Name</th> <th>Score</th> </tr> </thead> <tbody> <tr> <td>stima1</td> <td>13</td> </tr> <tr> <td>stima2</td> <td>8</td> </tr> <tr> <td>stima3</td> <td>5</td> </tr> </tbody> </table> <p>Peringkat 1</p>	Name	Score	stima1	13	stima2	8	stima3	5
Name	Score								
stima1	13								
stima2	8								
stima3	5								

3. Hasil Analisis

Algoritma alternatif tiga ini terbukti sebagai algoritma terbaik dibandingkan algoritma alternatif satu dan dua. Algoritma ini teroptimasi dalam pengambilan rute relatif terhadap keberadaan diamond, valuenya, dan teleporter, mempertimbangkan waktu dalam mencari diamond, dan juga aware terhadap keberadaan bot pemain lain yang terdapat di sekitar bot pemain.

3.4. Pemilihan Strategi Greedy

Dari keseluruhan alternatif algoritma greedy, didapatkan bahwa algoritma greedy alternatif tiga, terbukti dapat lebih mampu mengumpulkan diamonds dengan jumlah yang lebih besar dibandingkan dengan kedua alternatif lainnya. Algoritma greedy alternatif tiga juga terbukti dapat menghasilkan hasil yang jauh lebih konsisten dibandingkan kedua alternatif lainnya.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Algoritma Greedy pada Bot Diamonds

1. Algoritma get_direction()

Algoritma ini berfungsi untuk menentukan arah gerak bot, dengan mempertimbangkan posisi teleport dan bot di sekitar bot kita

```
function get_direction(Cur: posisi_bot, Dest: posisi_tujuan, tele:
array_teleports, bots: array_bots)-> Arah_gerak
{Menentukan arah gerak bot}
Deklarasi
  G: Arah_gerak
Algoritma:
  G.x = nilai tengah antara -1, 1, dan Dest.x-Cur.x
  G.y = nilai tengah antara -1, 1, dan Dest.y-Cur.y

  If (salah satu komponen G belum 0) then
    for bot in bots {traverse array bots}
      if (terdapat bot lawan di arah gerak horizontal) then
        G.y <- 0 {Tabrak bot lawan}
      Else if (terdapat bot lawan di arah gerak vertikal) then
        G.x <- 0 {Tabrak bot lawan}

  If (masih belum ada komponen G yang 0) then
    For t in tele
      If (terdapat teleport di arah gerak horizontal) then
        G.x <- 0 {Hindari dengan gerak vertikal dulu}
      Else If (terdapat teleport di arah gerak vertikal) then
        G.y <- 0 {Hindari dengan gerak horizontal dulu}

  If (masih belum ada komponen G yang 0) then
    G.x <- 0 {Gerak vertikal dulu}
  -> G
```

2. Algoritma bestRoute()

Algoritma ini untuk menentukan rute perjalanan yang paling optimal, dengan mempertimbangkan lokasi teleport di map.

```

function bestRoute(tele: array_teleport, dest: posisi_tujuan, pos:
posisi_sekarang) → (jarak_terkecil, tele_masuk, tele_keluar,
posisi_tujuan)
{Menentukan tujuan gerak optimal, dapat melalui teleport atau tidak
  Jika tidak melalui tele, tele_masuk = tele_keluar = posisi_tujuan}

Deklarasi
  bestDist: jarak_terkecil
  bestDest1: tele_masuk {Posisi}
  bestDest2: tele_keluar {Posisi}
  Function distTotal(dest: posisi_tujuan, pos: posisi_skrng) → jarak_manhattan
  {Mengembalikan jarak manhattan antara dua posisi}

Algoritma
  {Rute tanpa melalui teleport}
  bestDist ← distTotal(dest, pos)
  bestDest1 ← dest
  bestDest2 ← dest

  {Rute melalui teleport 1}
  dist ← distTotal(pos, tele[0]) + distTotal(tele[1], dest)
  if (dist < bestDist and pos != tele[0]) then
    bestDist ← dist
    bestDest1 ← tele[0]
    bestDest2 ← tele[1]

  {Rute melalui teleport 2}
  dist ← distTotal(pos, tele[1]) + distTotal(tele[0], dest)
  if (dist < bestDist and pos != tele[1]) then
    bestDist ← dist
    bestDest1 ← tele[1]
    bestDest2 ← tele[0]
  → (bestDist, bestDest1, bestDest2, dest)

  {[0] = jarak bot ke tujuan (bisa tele atau objek incaran)
  [1] = lokasi tujuan bot sekarang (bisa tele atau objek incaran)
  [2] = lokasi teleport keluar (optional)
  [3] = lokasi objek incaran }

```

3. Algoritma worthDim()

Algoritma ini untuk menentukan diamond lain dengan jarak/poin terkecil terhadap diamond pada posisi "pos".

```
function worthDim(teleports: array_teleport,diamonds:
array_diamonds,pos: posisi_diamond)→(jarak_diamond, poin_diamond)
{Mencari diamond dengan jarak/poin terkecil, dengan diamond pada lokasi
"pos"}
Deklarasi
  minDist: jarak_diamond
  Poin: poin_diamond
Algoritma
  {Inisialisasi}
    minDist ← 9999
    poin ← 1
    for d in diamonds
      dist ← bestRoute(teleports,d.position,pos)[0] {Jarak dengan d}
      if(dist/d.properties.points<minDist/poin and dist != 0) then
        minDist ← dist
        poin ← d.properties.points
    → (minDist,poin)
```

4. Algoritma bestRatio(), Taktik Greedy 1 (Greedy by Distance/Poin Ratio)

Algoritma ini untuk mengembalikan pilihan gerakan dengan nilai total jarak bagi total poin terkecil. Pilihan gerakan dapat berupa mengambil diamond pada dimPos saja, atau mengambil diamond pada dimPos dan kemudian lanjut mengambil diamond ke-dua hasil worthDim() terhadap diamond di dimPos.

```
Function bestRatio(poin: poin_diamond,dimPos: posisi_diamond, botPos:
posisi_bot, diamonds: array_diamonds, teleports: array_teleports)→
nilai_dist/poin
{Mengembalikan nilai terkecil total jarak dibagi total poin antara mengambil
diamond pada dimPos aja atau mengambil diamond dimPos dan diamond closest}

Deklarasi
  distDim: jarak_manhattan_bot_dengan_diamond
  Closest: informasi_diamond_terdekat_dengan_diamond_dimPos
Algoritma
  {distDim untuk mendapatkan jarak terdekat bot dengan diamond dimPos}
  distDim ← bestRoute(teleports,dimPos,botPos)[0]

  {closest untuk mendapatkan informasi diamond yang terdekat
  dengan dimPos}
```

```

closest ← worthDim(teleports,diamonds,dimPos)

{Mengembalikan nilai terkecil total jarak dibagi total poin,
 antara dua pilihan gerak tadi}
→min(distDim/poin, (distDim+closest[0])/(poin+closest[1]))

```

5. Algoritma bestGoal()

Algoritma ini berfungsi untuk menyeleksi diamond-diamond yang telah tersortir, dimulai dari diamond dengan hasil seleksi greedy terbaik, diseleksi berdasarkan kedekatan bot musuh ke diamond dibandingkan bot kita ke diamond. Jika terdapat bot musuh yang lebih dekat ke diamond dibandingkan kita, sementara kita pass. Tetapi jika semua diamond lebih dekat dengan bot musuh, kita ambil yang pertama (hasil seleksi greedy terbaik). Seleksi juga dilakukan jika diamond inventory berjumlah 4, kita akan melewati diamond merah (2 poin).

```

Function bestGoal(teleports: array_teleports,diamonds: array diamonds,bots:
array_bots,botPos: posisi_bot,currentDim: dim_bot)→tuple_lokasi_tujuan

{Fungsi seleksi, yang menyeleksi diamonds yang telah tersort dengan key sort
berupa hasil bestRatio masing2 diamond dari yang terkecil berdasarkan jarak
diamond dengan bot lawan}

Deklarasi
defDest: default_destination {indeks pertama array tersort}
Reachable: boolean_status_incar

Algoritma
for dims in diamonds
    {Jika jumlah diamond di inventory 4 dan diamond incaran berpoin 2,
     kita skip}
    if (currentDim=4 and dims.properties.points=2):
        continue
    reachable ← True
    ourBest ← bestRoute(teleports,dims.position,botPos)

    if (belum ada nilai default defDest):
        {Menentukan default tujuan, jika semua diamond memiliki bot lawan
         yang lebih dekat}
        defDest ← ourBest
    {Membandingkan jarak kita dan jarak lawan dengan diamondnya}
    for enemy in bots
        enemyBest ← bestRoute(teleports,dims.position,enemy.position)
        if (enemyBest[0]<ourBest[0]) then
            {Jika terdapat satu bot saja yang lebih dekat, langsung kita

```

```

pass}

        reachable ← False
        break
        {Jika bisa digapai (kita yang paling dekat dengan diamond),
        return diamond tersebut sebagai tujuan gerakan kita}
        if reachable then
            → ourBest
        {Return default jika tidak ada diamond yang memenuhi}
        → defDest

```

6. Algoritma betweenPoint()

Algoritma ini untuk mengecek apakah suatu point terletak di antara dua point lainnya.

```

function betweenPoint(point1: posisi, point2: posisi, point: posisi) →
boolean_kevalidan
{ Mengecek apakah point berada di antara point1 dan point2}

Deklarasi
Valid_x: kevalidan komponen x point
Valid_y: kevalidan komponen y point

Algoritma
valid_x ← (point1.x ≤ point.x ≤ point2.x) or (point2.x ≤ point.x ≤ point1.x)
valid_y ← (point1.y ≤ point.y ≤ point2.y) or (point2.y ≤ point.y ≤ point1.y)
→ valid_x and valid_y

```

7. Algoritma extraMove()

Algoritma ini untuk menghitung jumlah gerakan tambahan yang dibutuhkan untuk mencapai diamond, yang berlawanan dengan arah gerakan ketika ke base. Misalnya jika arah untuk ke base adalah kanan bawah, maka menghitung gerakan tambahan kiri atas yang diperlukan untuk ke diamond

```

Function extraMove(dest: posisi_tujuan, pos: posisi_sekarang, dimPos:
posisi_bot) → jumlah_extramove

{Menghitung jumlah gerakan tambahan yang dibutuhkan untuk mencapai diamond,
yang berlawanan dengan arah gerakan ketika ke base}

Deklarasi
    upperY, upperX, lowerY, lowerX: integer

```

```

    Temp: integer
Algoritma
    if betweenPoint(dest,pos,dimPos) then
        return 0
    temp ← 0
    upperY ← max(dest.y,pos.y)
    lowerY ← min(dest.y,pos.y)
    upperX ← max(dest.x,pos.x)
    lowerX ← min(dest.x,pos.x)
    if dimPos.y>upperY:
        Temp ← temp+dimPos.y-upperY
    elif dimPos.y<lowerY:
        Temp ← temp+lowerY-dimPos.y
    if dimPos.x>upperX:
        Temp ← temp+dimPos.x-upperX
    elif dimPos.x<lowerX:
        Temp ← temp+lowerX-dimPos.x
    → temp

```

8. Algoritma totalDistTravel(), Taktik Greedy 2 (Greedy by total distance from bot to diamond and diamond to base)

Algoritma ini untuk menghitung total perjalanan dari posisi bot sekarang ke diamond, ditambah dari diamond ke base, dengan mempertimbangkan teleport.

```

Function totalDistTravel(teleports: array_teleports,botPos:
posisi_bot,dimPos: posisi_diamond,base: posisi_base) → total_jarak
{Total jarak perjalanan dari posisi sekarang ke diamond dan kemudian ke
base}
Deklarasi
    botToTele,teleToDim,dimToTele,teleToBase: tuple_info_gerak
Algoritma
    {botToTele bernilai jarak bot ke teleport, jika tidak lewat
    teleport jarakny lgsg ke diamond}
    botToTele ← bestRoute(teleports,dimPos,botPos)

    {teleToDim, jarak teleport keluar ke diamond, jika tidak
    lewat teleport bernilai 0}
    teleToDim ← distTotal(dimPos,botToTele[2])

    {dimToTele bernilai jarak diamond ke teleport, jika tidak lewat
    teleport jarakny lgsg ke base}

```

```

dimToTele ← bestRoute(teleports,base,dimPos)

{teleToDim, jarak teleport keluar ke diamond, jika tidak
 lewat teleport bernilai 0}
teleToBase ← distTotal(base,dimToTele[2])
→ botToTele[0]+teleToDim+dimToTele[0]+teleToBase

```

9. Program Utama

Pada program utama, kami menggunakan dua taktik greedy, yang disesuaikan pemakaiannya berdasarkan jumlah diamond di inventory dan waktu bermain yang tersisa. Berikut penjelasan lebih lengkap:

- Jika jumlah diamond di inventory adalah 5, kembali ke base
- Jika diamond di inventory lebih kecil dari 3 dan sisa waktu bermain masih lebih besar dari 20, gunakan taktik Greedy 1 (Greedy by Distance/Poin Ratio)
- Jika jumlah diamond di inventory lebih besar sama dengan 3, gunakan taktik Greedy 2 (Greedy by total distance from bot to diamond and diamond to base). Jika diamond incaran tidak sejalan balik ke base dan extra move untuk diamond lebih besar dari 2, balek ke base.
- Jika jarak ke tujuan lebih besar sama dengan jarak ke shuffle ditambah 5 dan diamond di inventory lebih kecil dari 3, tekan tombol shuffle
- Jika base berada di antara bot kita dan tujuan, dan posisi bot bukanlah posisi base, kita kembali ke base dulu
- Jika total perjalanan bot ke diamond dan kembali ke base, lebih besar dari jumlah waktu yang tersisa (waktu tiap gerakan $\approx 1s$) dan diamond di inventory > 0 , balek ke base
- Kemudian menggunakan `get_direction()` untuk menentukan arah gerakan bot, dengan melihat letak bot dan teleport di sekitar

```

Function next_move(self, board_bot: GameObject, board: Board)-> arah_gerak
{ Program utama untuk menentukan gerakan bot }
Algoritma
    {Setup array bots, teleports, diamonds, dan lokasi shuffle}
    {teleports : List of position}
    teleports = []
    {bots : List of Game Object}
    bots = []
    {diamonds : List of Game Object}
    diamonds = []
    {Position dimana shuffle berada}
    shuffle = 0
    for x in board.game_objects:

```

```

        if(x.type == "BotGameObject"):
            bots.append(x)
        elif x.type == "DiamondGameObject":
            diamonds.append(x)
        elif x.type == "TeleportGameObject":
            teleports.append(x.position)
        elif x.type == "DiamondButtonGameObject":
            shuffle = x.position

    {Variabel-variabel pendukung}
    props = board_bot.properties #Properties dari bot kita
    botPos = board_bot.position #Posisi bot kita sekarang
    base = props.base #Posisi base
    toBase = bestRoute(teleports,base,botPos)
    {Rute menuju base dari lokasi sekarang}

    {Jika jumlah diamond di inventory adalah 5, kita balik ke base}
    if props.diamonds==5 then
        self.goal = toBase
    {Jika jumlah diamond di inventory kurang dari 3 dan waktu permainan
yang
        tersisa lebih dari 20 detik}
    Else if props.diamonds<3 and (props.milliseconds_left//1000>=20):
        diamonds = hasil sort diamond berdasarkan teknik greedy 1, greedy
by
        distance/value ratio
        {Seleksi kandidat diamonds}
        self.goal =
bestGoal(teleports,diamonds,bots,botPos,props.diamonds)

    {Jika jumlah diamond di inventory lebih dari sama dengan 3 atau sisa
waktu permainan kurang dari 20 detik lagi}
    else:
        diamonds = hasil sort diamonds berdasarkan taktik greedy 2 (total
distance), jika ada yang sama disortir berdasarkan jarak terdekat
dgn bot

        {Seleksi kandidat diamonds}
        self.goal =
bestGoal(teleports,diamonds,bots,botPos,props.diamonds)

    {Jika jumlah diamond inventory lebih besar sama dengan 3}
    if props.diamonds>=3 then
        if (diamond tidak sejalan balik ke base dan extraMove>2) then
            self.goal = toBase

```

```

        shuffleRoute = bestRoute(teleports,shuffle,botPos)
        if Jika jarak kita dengan tujuan lebih besar sama dengan jarak kita
        ke
            shuffle ditambah 5 and props.diamonds<3:
                self.goal = shuffleRoute

        if Jika base berada di antara bot kita dan tujuan dan posisi bot
        bukanlah posisi base then
            self.goal = toBase

        # Jika waktu untuk mengambil diamond dan kembali ke base tidak cukup
        (lebih kecil dari sisa waktu), dan terdapat diamond di inventory, kembali ke
        base
        if Jika waktu untuk mengambil diamond dan kembali ke base tidak cukup
        dan terdapat diamond di inventory then
            self.goal = toBase

        delta_x,delta_y = get_direction(
            botPos.x,
            botPos.y,
            self.goal[1].x,
            self.goal[1].y,
            teleports,
            bots,
        )
        return delta_x, delta_y

```

4.2. Struktur Data

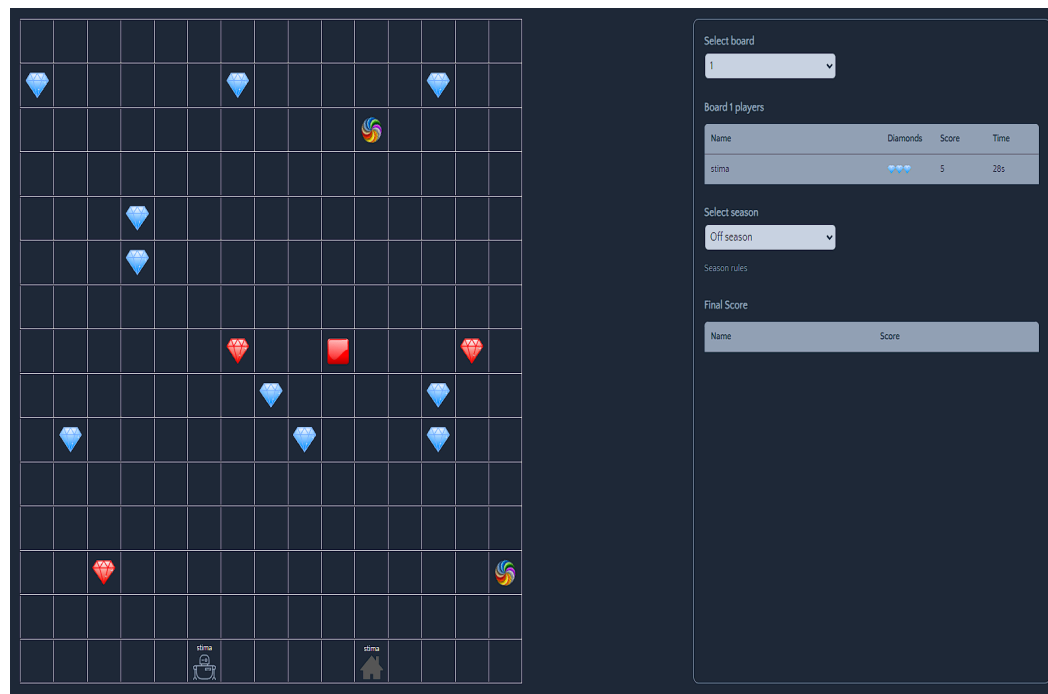
Dalam program ini, data serta struktur data yang digunakan adalah sebagai berikut.

- Teleports: List dari posisi teleport (List of Positions)
- Bots: List dari bots (List of Game Object)
- Diamonds: List dari diamond-diamond (List of Game Object)
- Shuffle: Posisi dari tombol shuffle (Position)
- Props: properties dari bot kita (properties)
- botPos: Posisi dari bot kita (position)
- Base: Posisi dari base (Base)
- toBase: tuple yang berisi informasi perjalanan dari posisi bot sekarang ke base. Tuple berbentuk (integer,position,position,position)
- Self.goal: tuple yang berisi informasi perjalanan dari posisi bot sekarang ke posisi tujuan. Tuple berbentuk (integer,position,position,position)

4.3. Pengujian dan Analisis Desain

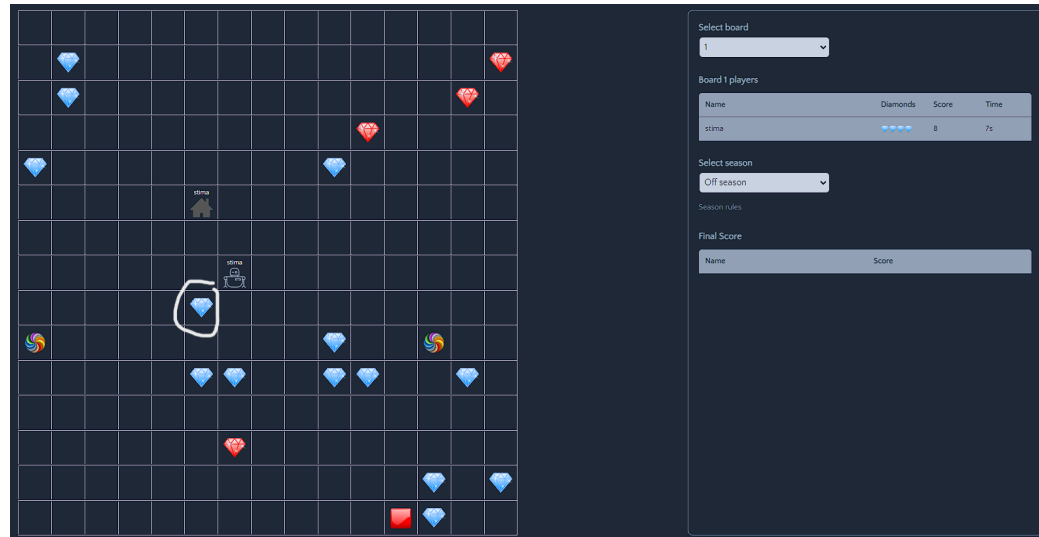
Kami melakukan analisis dan pengujian dengan cara mempertarungkan bot yang kami punya dengan bot-bot alternatif yang lain. Setelah itu kami amati perilaku (*Behaviour*) dari bot tersebut dengan menggunakan *visualizer* yang tersedia pada game Diamonds. Berikut adalah pengujian-pengujian yang kami lakukan

1. Kasus 1



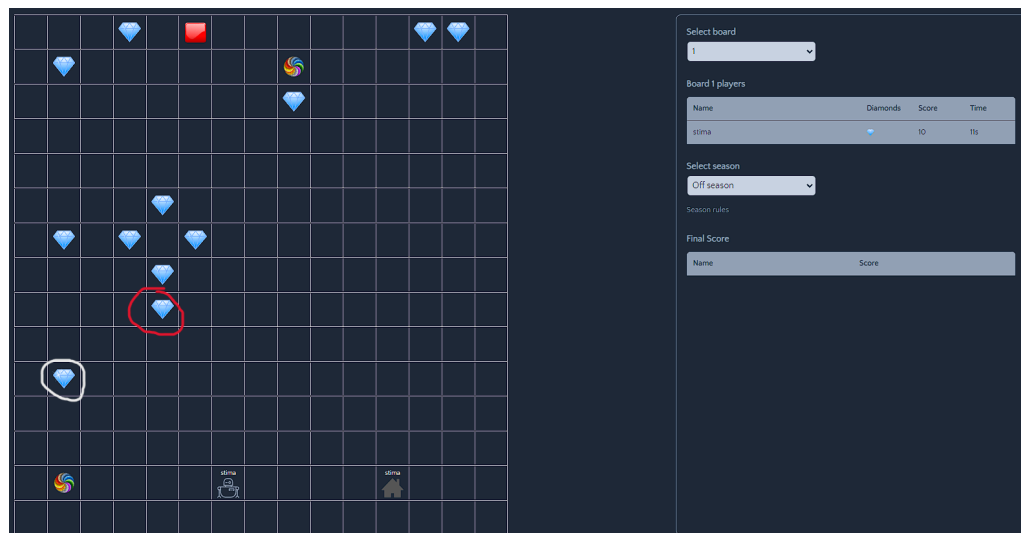
Kasus pertama, diamond di inventory berjumlah 3 dan posisi bot seperti gambar di atas. Dengan algoritma greedy kami, bot akan menerapkan teknik *Greedy by Total Distance Bot-Diamond-Base*, yang akan aktif ketika jumlah diamond di inventory lebih besar sama dengan tiga. Bot pada kasus ini, akan mengabaikan bot merah di kirinya dan akan pulang ke base. Hal ini karena extra move yang diperlukan untuk ke diamond merah tersebut sebesar 5, yakni ke kiri sebanyak tiga kali dan ke atas sebanyak dua kali. Sedangkan dalam program kami dibatasi paling banyak 1, kalau diamond tersebut mau diambil. Menurut kami, keputusan ini yang optimal karena kita mengamankan diamonds tersebut dahulu dari resiko dicuri bot lain, karena jarak diamond merah tersebut dengan base relatif jauh.

2. Kasus 2



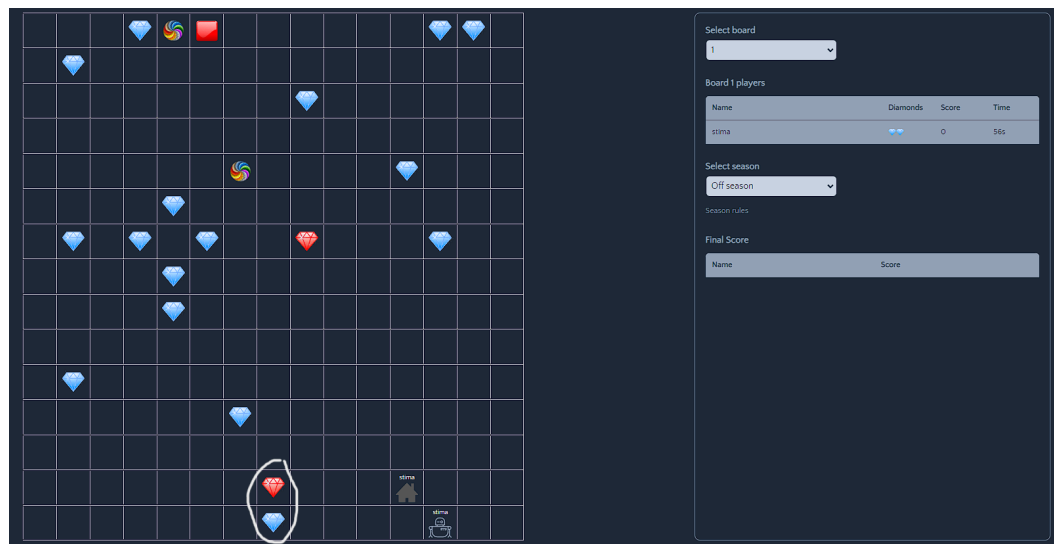
Kasus ke-dua, diamond di inventory bot sebanyak 4 dan posisi bot seperti gambar di atas. Bot akan bergerak dengan teknik *Greedy by Total Distance Bot-Diamond-Base*, yang aktif karena jumlah diamond di inventory ≥ 3 . Bot akan mengambil diamond yang dilingkari putih karena bot tersebut memiliki total jarak travel paling kecil ($2 \text{ (Bot-Diamond)} + 3 \text{ (Diamond-Base)} = 5$). Walaupun diamond tersebut tidak sejalan pulang, bot tetap akan mengambil nya karena extra move yg diperlukan hanya 1, yakni satu langkah ke bawah. Menurut kami, ini menjadi solusi yang optimal karena inventory bot sekarang 4 diamond dan untuk mengambil diamond tersebut hanya diperlukan satu extra move saja.

3. Kasus 3



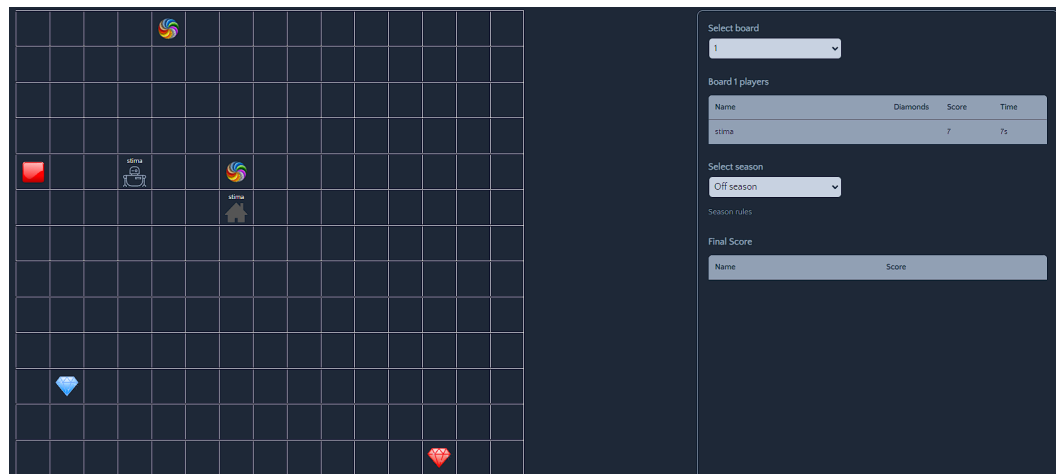
Kasus ke-tiga, diamond di inventory berjumlah 1 dan waktu tersisa 10 detik. Dengan sisa waktu segini, diamond akan menjalankan teknik *Greedy by Total Distance Bot-Diamond-Base*. Program akan memilih diamond yang dilingkari merah sebagai incarannya, dengan total travel distance bot-dim-base terkecil sebanyak 19, dibandingkan dengan yg warna putih sebanyak 21. Akan tetapi, karena waktu yang tersisa tidak cukup untuk bot mengambil diamond dan kembali ke base ($19 > 10$) dan diamond di inventory sekarang > 0 , bot akan kembali ke base untuk mengamankan poinnya. Menurut kami, ini menjadi solusi yang optimal karena jika kita pergi mengambil diamond tersebut, perjalanan kita akan menjadi sia-sia karena waktu yang tersisa tidak cukup bagi kita untuk mengamankan poinnya. Dan juga, kita lebih baik mengamankan diamond di inventory sebagai poin.

4. Kasus 4



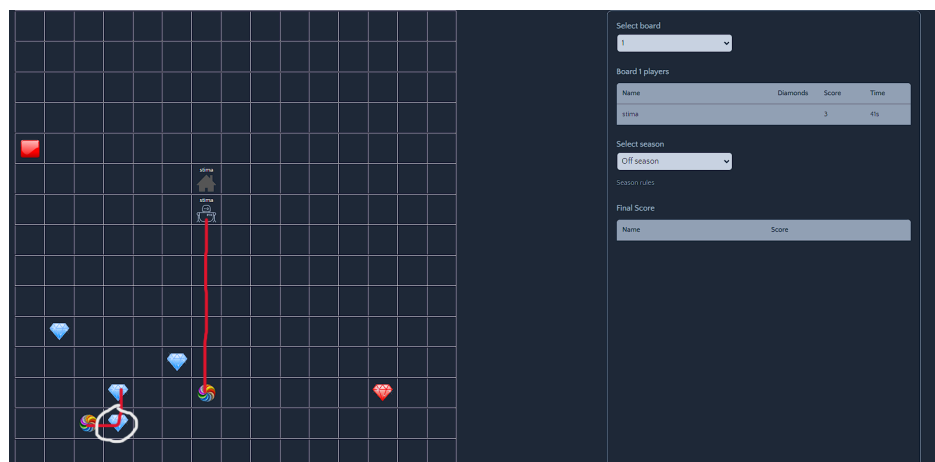
Kasus berikutnya, diamond di inventory sebanyak 2 dan sisa waktu masih 56 detik. Dengan situasi begini, bot akan menjalankan teknik *Greedy by Distance/Value ratio*. Perhatikan diamonds yang dilingkari putih. Diamond merah akan memiliki best ratio $7/3$, didapat dari mengambil diamond merah kemudian mengambil diamond biru dibawahnya. Sedangkan diamond biru, akan memiliki best ratio $6/3$, didapat dari mengambil diamond biru terlebih dahulu, baru mengambil diamond merah. Karena pengambilan diamond biru memiliki ratio lebih kecil, maka diamond tersebut menjadi tujuan kita. Karena base tidak berada sejalan dengan arah jalan kita, maka kita tidak mengunjungi base.

5. Kasus 5



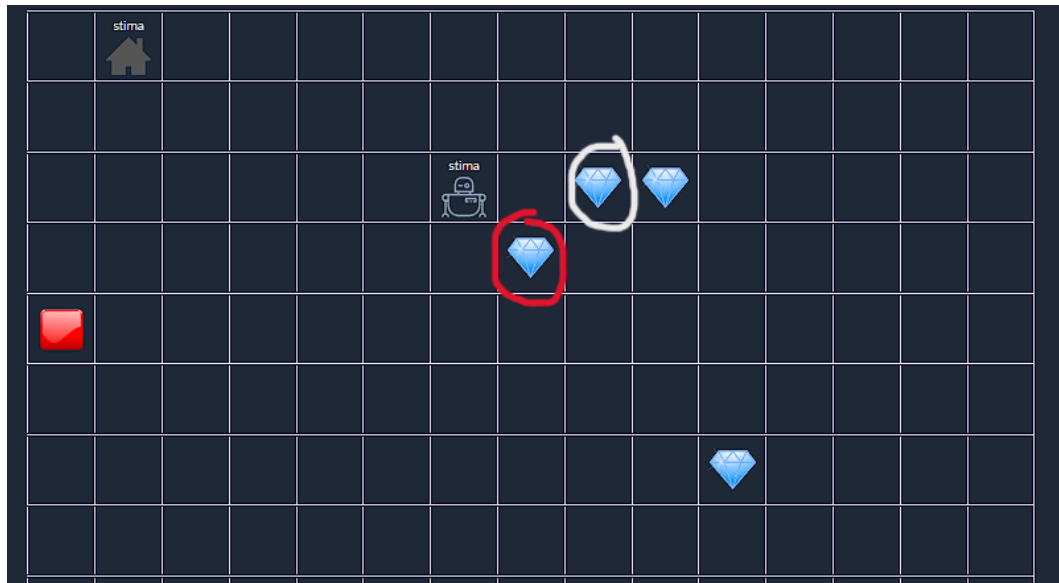
Kasus berikutnya, inventory kosong dan waktu yang tersisa 7 detik. Bot akan menerapkan teknik *Greedy by Total Distance Bot-Diamond-Base* karena waktu sisa lebih kecil dari 20 detik. Diamond dengan total distance terkecil dimiliki diamond biru dengan jumlah step 19, dibandingkan dengan diamond merah yang 30 step. Akan tetapi, jumlah step yang diperlukan bot untuk ke tombol shuffle ditambah 5 ($3+5= 8$ step) lebih besar sama dengan jumlah step dari bot ke diamond (8 step) sehingga bot akan lebih memilih untuk menekan tombol shuffle. Menurut kami, ini menjadi solusi yang optimal karena kami rasa tidak *worth* bagi kita berjalan jauh hanya untuk satu atau dua diamond saja. Lebih baik kita meng-*shuffle* diamond, dengan kemungkinan akan ada diamonds yang terspawn dekat kita atau base.

6. Kasus 6



Kasus berikutnya, inventory bot kosong dan waktu yang tersisa masih 41 detik. Bot akan menerapkan teknik *Greedy by Distance/Value Ratio* karena diamond di inventory < 3 dan waktu tersisa masih ≥ 20 detik. Bot akan mengambil diamond yang dilingkari putih dengan best rasionya $8/2$, didapat dari mengambil diamond tersebut dan mengambil diamond di atasnya. Bot akan mengambil jalur garis merah melewati teleport karena memperkecil jarak bot dengan diamond. Kami rasa ini sudah menjadi solusi yang optimal, karena hanya dengan berjalan 8 langkah, kita sudah mendapat 2 poin. Ini lebih baik dibandingkan jika kita berjalan ke diamond lain.

7. Kasus 7



Kasus berikutnya, dengan inventory bot lebih kecil dari 3 dan waktu tersisa masih lebih besar dari 20 detik. Disini, bot akan menjalankan teknik *Greedy by Distance/Value Ratio*. Diamond putih memiliki rasio $3/2$, didapat dengan mengambil diamond tersebut dan mengambil diamond di kanannya. Diamond yang dilingkari merah memiliki rasio $2/1$, didapat dengan mengambil diamond tersebut saja. Sehingga, bot akan memilih diamond putih karena rasio jarak per value poinnya yang paling kecil.

Terdapat beberapa skenario dimana bot kami memberikan solusi yang kurang optimal, berikut skenario-skenarionya.

1. Skenario 1



Pada skenario ini, bot ingin kembali ke base, tetapi sebuah teleport menghalangi jalannya. Bot kami, akan masuk ke dalam teleport, keluar di teleport yang lain, masuk lagi ke teleport lain, keluar di teleport awal dan masuk ke base. Walaupun jumlah step yang dibutuhkan sama jika kita mengelilingi teleport lewat atas atau bawah (sama-sama 4 step), tetapi langkah kami memiliki resiko. Resikonya jika setelah masuk teleport, letak teleport akan ter-*refresh* dan posisi kita jauh dari base.

2. Skenario 2



Pada skenario ini, jika bot kami adalah bot kiri, dan kita ingin bergerak ke kanan, bot kami akan tetap bergerak ke kanan tanpa memperhatikan bot lawan di depannya. Hal ini berisiko untuk ditabrak, karena jika kita yang bergerak duluan, bot musuh bisa mendeteksi kita dan berisiko untuk ditabrak. Hal ini akan merugikan kita dan diamond di inventory kita akan dicuri oleh lawan.

BAB 5

PENUTUP

5.1. Kesimpulan

Setelah melakukan berbagai eksplorasi terkait algoritma greedy dan implementasinya pada permainan Diamonds. Kami dapat membangun sebuah bot sangat optimal untuk permainan ini. Akan tetapi, algoritma *greedy* yang kami gunakan tidak dapat selalu memenangkan permainan ini, hal ini dikarenakan oleh dua faktor utama.

Faktor yang pertama adalah terdapat algoritma *greedy* lain yang lebih baik dalam kondisi tertentu dibandingkan punya kami sehingga dapat menghasilkan hasil yang lebih optimal dari algoritma *greedy* kami. Faktor yang kedua adalah faktor *spawnpoint* dalam setiap round. Posisi awal dalam permainan ini menjadi salah satu faktor penting dalam memenangkan permainan. Sebuah *spawnpoint* yang terletak dalam daerah yang memiliki konsentrasi diamonds banyak akan mendapatkan keuntungan yang sangat besar, hal ini dikarenakan jarak yang sangat dekat dalam mengambil diamond dan membawanya ke base secara utuh. Hal ini adalah hal yang di luar kontrol algoritma *greedy*, dikarenakan seluruh hal ini diatur secara random.

Akan tetapi, algoritma yang kami gunakan sudah memenuhi elemen-elemen dari algoritma *greedy*. Dengan kemampuan algoritma greedy pada umumnya yang tidak selalu menghasilkan hasil yang optimal, algoritma greedy yang telah kami implementasikan benar dan cukup optimal. Algoritma yang kami gunakan cukup untuk mendapatkan poin yang cukup besar untuk waktu yang diberikan, dan memenangkan permainan walaupun tidak setiap permainan. Dengan baiknya performa yang telah dicapai oleh algoritma greedy kami, kami yakin algoritma greedy ini sudah cukup baik, dan walaupun demikian, masih terdapat algoritma greedy lainnya yang dapat mendapatkan hasil yang lebih baik dan lebih konsisten dibandingkan dengan algoritma greedy kami.

5.2. Saran

Kami berharap adanya pendalaman atau riset mengenai algoritma yang paling efisien dalam menyelesaikan permasalahan dalam permainan Diamonds, agar kami dapat memiliki bahan yang dapat digunakan sebagai tolak ukur dalam mengerjakan algoritma yang dibatasi dengan pendekatan *greedy* ini.

5.3. Komentar dan Refleksi

Kami bangga terhadap kelompok kami yang telah berusaha bersama - sama untuk mendapatkan algoritma greedy yang terbaik dan berhasil menyelesaikan tugas besar ini. Kami berterima kasih kepada bapak Rinaldi Munir yang telah mengajarkan kami dasar - dasar algoritma greedy sehingga dapat mengimplementasikannya dalam pembuatan logic bot dalam tugas besar ini.

Dengan tugas besar ini, kami juga mempelajari banyak hal mengenai strategi greedy dan implementasi dari berbagai variasi pendekatan algoritma greedy. Kami juga dapat belajar sedikit demi sedikit mengenai cara kerja dari bot engine. Kami berharap ilmu yang kami dapatkan dari tugas besar ini dapat bermanfaat dan membantu kami dalam pengerjaan tugas selanjutnya, dan menggunakannya di masa yang akan datang.

LAMPIRAN

LINK REPOSITORY

- [Repository Game Engine](#)
- [Link repository GitHub](#)

LINK VIDEO YOUTUBE

- [Link Video Youtube](#)

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522029	Ignatius Jhon Hezkiel Chan	<i>Algoritma, video, dan laporan dibagi rata</i>
13522043	Daniel Mulia Putra Manurung	
13522093	Matthew Vladimir Hutabarat	

DAFTAR PUSTAKA

YouTube: Home, 9 November 2017,

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm>.

Accessed 9 March 2024.

“Python Tutorial.” *W3Schools*, <https://www.w3schools.com/python/default.asp>. Accessed 9 March 2024.

Spesifikasi Tugas Besar 1 IF2211 Pemanfaatan Greedy dalam pembuatan bot permainan Diamonds. (2024). Bandung: Institut Teknologi Bandung.