

LAPORAN TUGAS BESAR
IF2211 - STRATEGI ALGORITMA

*Pemanfaatan Algoritma Greedy dalam Pembuatan Bot Permainan
Diamonds*

Kelas Mahasiswa K01

Dosen Pengampu:

Dr. Ir. Rinaldi, M.T.



Disusun Oleh:
Kelompok 40 - KOIN/ONDO

Mesach Harmasendo 13522117
Enrique Yanuar 13522077
Juan Alfred Wijaya 13522073

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2024

DAFTAR ISI

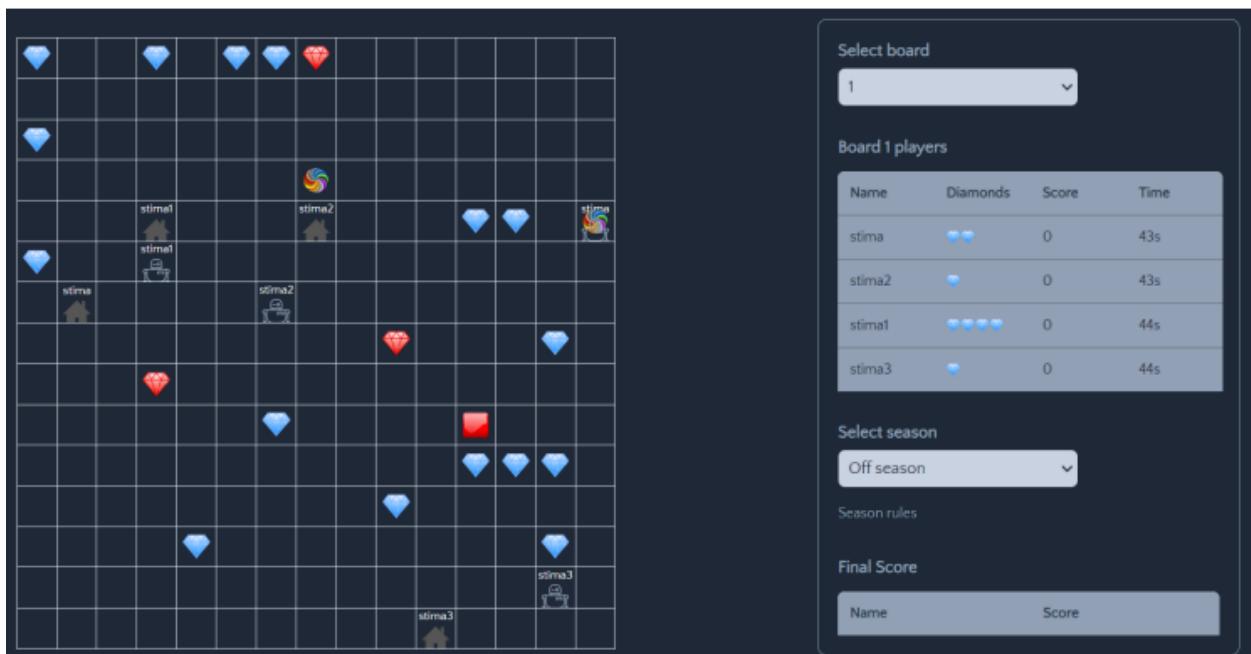
DAFTAR ISI.....	2
BAB 1 DESKRIPSI TUGAS.....	4
BAB 2 LANDASAN TEORI.....	8
2.1 Algoritma Greedy.....	8
2.2 Game Engine Diamonds.....	9
2.3 Bot Starter Pack.....	10
2.4 Langkah untuk Menjalankan Game Diamonds.....	10
BAB 3 APLIKASI STRATEGI GREEDY.....	15
3.1 Pemilihan Fungsi Kelayakan.....	15
3.2 Alternatif Algoritma Greedy.....	17
3.2.1 Alternatif Greedy by Shortest to Bot.....	17
3.2.2 Alternatif Greedy by Shortest to Base.....	19
3.2.3 Alternatif Greedy by Shortest To Bot Base.....	21
3.2.4 Alternatif Greedy by Shortest to Highest Reward.....	24
3.2.5 Alternatif Greedy by Shortest to Highest Block.....	26
3.2.6 Alternatif Greedy by Highest Block Reward per Distance from Bot.....	29
3.2.7 Alternatif Greedy by Highest Block Reward per Distance from Base.....	32
3.2.8 Alternatif Greedy by Highest Block Reward per Distance from Bot and Base.....	35
3.2.9 Alternatif Greedy by Highest Reward Per Distance From Bot.....	38
3.2.10 Alternatif Greedy by Highest Reward Per Distance From Bot and Base.....	40
3.2.11 Alternatif Greedy by Highest Reward Per Distance From Base.....	43
3.2.12 Alternatif Greedy by Shortest to Bot V2.....	45
3.3 Strategi Greedy yang Dipilih.....	47
BAB 4 IMPLEMENTASI DAN PENGUJIAN.....	52
4.1 Implementasi Algoritma Greedy yang Dipilih.....	52
4.2 Struktur Data Program.....	54
4.3 Analisis dan Pengujian.....	57
4.3.1 Kasus Algoritma Utama.....	57
4.3.2 Kasus Efektif.....	59
4.3.3 Kasus Tidak Efektif.....	60
4.3.4 Kasus Balik ke Base Berdasarkan Waktu.....	62
4.3.5 Kasus Balik ke Base Kapasitas Penuh.....	63
4.3.6 Kasus Balik ke Base Kapasitas Sudah 4 Diamond dan Target Selanjutnya Diamond Merah.....	64
4.3.7 Kasus Balik ke Base Karena Jarak ke Base Lebih Dekat dari Target Selanjutnya.....	65
4.3.8 Kasus Mengambil Diamond Red Button.....	66
4.3.9 Kasus Menggunakan Teleporter.....	67

BAB 5 KESIMPULAN DAN SARAN.....	69
5.1 Kesimpulan.....	69
5.2 Saran.....	70
LAMPIRAN.....	71
DAFTAR PUSTAKA.....	72

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



Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi *greedy* dalam membuat bot ini. Program permainan *Diamonds* terdiri atas:

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

- a. Program untuk memanggil API yang tersedia pada backend
- b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma *greedy* untuk bot kelompok kalian)
- c. Program utama (main) dan utilitas lainnya

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

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

<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

Komponen-komponen dari permainan *Diamonds* antara lain:

1. *Diamonds*

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

2. Red Button/Diamond Button

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

3. Teleporters

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

4. Bots and Bases

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

5. *Inventory*

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

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

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

BAB 2

LANDASAN TEORI

2.1 Algoritma *Greedy*

Algoritma Greedy adalah sebuah metode algoritmik yang membangun solusi langkah demi langkah, selalu memilih langkah yang tampaknya paling optimal pada saat itu. Dengan kata lain, algoritma ini membuat keputusan yang optimal secara lokal dengan harapan mendapatkan solusi yang optimal secara global. Algoritma Greedy tidak selalu menghasilkan solusi terbaik, tetapi sering digunakan karena kecepatannya dan kesederhanaannya.

Prinsip dasar dari algoritma greedy adalah memilih elemen yang terbaik atau paling menguntungkan pada setiap langkah tanpa mempertimbangkan konsekuensi di masa depan. Dalam banyak kasus, algoritma greedy membuat keputusan berdasarkan informasi yang tersedia pada saat itu, tanpa mempertimbangkan pilihan yang mungkin muncul di kemudian hari.

Elemen-elemen dalam algoritma greedy umumnya meliputi:

1. **Himpunan Kandidat (C):** Ini adalah kumpulan semua elemen yang berpotensi menjadi bagian dari solusi. Dalam konteks masalah praktis, ini bisa berupa daftar barang yang bisa dipilih, rute yang bisa diambil, atau pilihan lain yang relevan dengan masalah yang dihadapi.
2. **Himpunan Solusi (S):** Ini adalah kumpulan elemen yang telah dipilih dan dianggap sebagai bagian dari solusi yang sedang dibangun. Awalnya, himpunan ini kosong dan elemen ditambahkan ke dalamnya berdasarkan kriteria tertentu hingga terbentuk solusi yang dianggap optimal atau memenuhi syarat tertentu.
3. **Fungsi Solusi:** Fungsi ini digunakan untuk mengevaluasi apakah himpunan solusi (S) yang telah dibentuk sudah memenuhi syarat sebagai solusi yang valid atau belum. Dalam beberapa kasus, ini bisa berarti memeriksa apakah semua syarat telah terpenuhi atau apakah tujuan telah dicapai.
4. **Fungsi Seleksi:** Fungsi ini bertanggung jawab untuk memilih elemen dari himpunan kandidat (C) yang akan ditambahkan ke dalam himpunan solusi (S). Pemilihan ini dilakukan berdasarkan strategi greedy, yang biasanya mengutamakan elemen yang memberikan keuntungan terbesar atau biaya terendah pada saat itu.

5. **Fungsi Kelayakan:** Setelah elemen terpilih melalui fungsi seleksi, fungsi kelayakan digunakan untuk memeriksa apakah elemen tersebut layak untuk dimasukkan ke dalam himpunan solusi (S). Hal ini melibatkan pemeriksaan terhadap berbagai batasan atau syarat yang harus dipenuhi oleh solusi.
6. **Fungsi Objektif:** Fungsi ini digunakan untuk mengevaluasi seberapa baik himpunan solusi (S) yang telah dibentuk. Dalam konteks optimisasi, fungsi objektif ini bisa berupa memaksimalkan nilai atau keuntungan, atau meminimalkan biaya atau kerugian.

Dengan menggunakan elemen-elemen tersebut, maka dapat dipastikan bahwa algoritma *greedy* melibatkan pencarian sebuah himpunan bagian S , dari himpunan kandidat C . Maka dari itu, S harus memenuhi beberapa kriteria yang telah ditentukan melalui pemeriksaan fungsi kelayakan, yaitu S menyatakan suatu solusi dan S dioptimasi dengan menggunakan fungsi objektif.

2.2 Game Engine Diamonds

Game Engine Diamonds adalah salah satu *starter pack* yang diperlukan untuk memainkan dan menjalankan *game diamonds* ini. *Game Engine* ini merupakan aplikasi berbasis *web* yang dibuat dengan menggunakan docker sehingga lebih mudah untuk dijalankan di berbagai komputer ataupun laptop. Untuk menjalankan program ini hanya perlu mem-*build* container berdasarkan *docker image* yang sudah ada kemudian hanya tinggal menjalankan container yang sudah di build tersebut. Aplikasi *game diamonds* ini terdiri dari dua bagian utama yaitu frontend dan juga backend. Aplikasi ini dibuat dengan menggunakan bahasa JavaScript dengan menggunakan framework React.js untuk frontend dan framework Nest.js untuk backend. Untuk memainkan permainan ini user perlu meng-*install* *game engine* ini melalui link berikut: <https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>

Penjelasan untuk komponen-komponen penting yang terdapat dalam game engine adalah sebagai berikut:

- a. *Backend*: *Backend* merupakan bagian dari game engine yang bertugas untuk mengatur segala keperluan yang berhubungan dengan *game diamonds* ini dari sisi server. *Backend* akan mengatur logika dari permainan ini secara keseluruhan. *Backend* juga akan menyediakan berbagai macam *endpoint* yang bisa digunakan baik oleh *frontend* maupun oleh *bot starter pack* yang berisi algoritma robot yang telah kita buat. *Backend* juga bertugas untuk

melakukan pengelolaan basis data untuk menyimpan data-data permainan seperti *highscore* ataupun *config board* yang akan digunakan dalam permainan. Melalui *backend* ini pula kita bisa memainkan permainan ini dengan algoritma robot yang telah kita buat dan robot kita akan terlihat di *frontend* dari permainan ini.

- b. *Frontend*: *Frontend* merupakan bagian dari *game engine* yang berinteraksi langsung dengan pengguna. *Frontend* ini juga bertugas untuk menampilkan antarmuka dari permainan ini agar bisa dilihat oleh pengguna. *Frontend* akan mengatur segala hal yang berhubungan dengan tampilan visual dari permainan ini dan mengatur bagaimana pengguna bisa berinteraksi dengan permainan melalui antarmuka yang sudah dibuat.

2.3 Bot Starter Pack

Bot Starter Pack merupakan starter pack yang diperlukan untuk membuat algoritma robot untuk memainkan permainan ini. *Bot Starter Pack* ini dibuat dengan menggunakan bahasa pemrograman *python*. *Starter Pack* ini berisi berbagai macam hal yang diperlukan untuk membangun dan mengembangkan bot yang nantinya akan digunakan dalam permainan. Bagian inti dari *Starter Pack* ini tentu saja adalah algoritma greedy yang dibuat untuk menggerakan suatu bot agar bisa memenangkan permainan. Algoritma ini akan dibuat pada folder “./game/logic” yang terdapat dalam program *Starter Pack* ini. *Starter Pack* ini juga sudah menyiapkan segala macam keperluan agar bot yang telah dibuat dengan suatu algoritma nantinya bisa terhubung dengan *game engine* dan masuk kedalam permainan. Ketika program ini dijalankan maka program ini akan secara otomatis memanggil *endpoint* yang telah disediakan oleh *game engine* untuk mendaftarkan bot yang telah dibuat sehingga bisa masuk ke dalam permainan dan memainkan permainan ini. Untuk *Starter Pack* ini bisa di-*download* pada link berikut: <https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

2.4 Langkah untuk Menjalankan Game Diamonds

Langkah pertama yang harus dilakukan untuk dapat menjalankan permainan ini tentu saja adalah men-*download* *game engine* dan *bot starter pack* terlebih dahulu. Pastikan juga sudah meng-*install requirement* yang diperlukan seperti Node.js, Docker desktop, Python, dan juga Yarn. Jika semua keperluan sudah di *download* maka langkah-langkah untuk menjalankan program akan dijelaskan sebagai berikut:

a. Langkah menjalankan *game engine*:

1. Extract zip *game engine* yang telah di-download, kemudian masuk ke folder hasil ekstraksinya dan buka terminal
2. Masuk ke root directory dari project

```
cd tubes1-IF2110-game-engine-1.1.0
```

3. Install dependencies menggunakan Yarn

```
yarn
```

4. Setup default environment variable dengan menjalankan script berikut

Untuk Windows

```
./scripts/copy-env.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/copy-env.sh  
./scripts/copy-env.sh
```

5. Setup local database, pastikan sudah menjalankan *docker desktop* terlebih dahulu kemudian jalankan perintah berikut di terminal

```
docker compose up -d database
```

Lalu jalankan script berikut. Untuk Windows

```
./scripts/setup-db-prisma.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/setup-db-prisma.sh
```

```
./scripts/setup-db-prisma.sh
```

6. Build program dengan menjalankan perintah berikut

```
npm run build
```

7. Run program dengan perintah berikut

```
npm run start
```

Jika program berhasil dijalankan maka akan terlihat tampilan seperti dibawah ini pada Terminal

```
[0] [nodemon] to restart at any time, enter `rs`  
[0] [nodemon] watching dir(s): dist\**\*.env  
[0] [nodemon] watching extensions: ts, map, js, json  
[0] [nodemon] starting 'nest start'  
[0] [Nest] 3476 - 02/15/2024, 10:39:58 PM LOG [NestFactory] Starting Nest application...  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [InstanceLoader] AppModule dependencies initialized +106ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RoutesResolver] BoardsController {/api/boards}: +100ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/boards, GET} route +3ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/boards/:id, GET} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/bots, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/bots/recover, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/bots/:id/join, POST} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/bots/:id/move, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RoutesResolver] HighscoresController {/api/highscores}: +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/highscores/:seasonId, GET} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RoutesResolver] RecordingsController {/api/recordings}: +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/recordings/:seasonId, GET} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/recordings/:id, GET} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RoutesResolver] SeasonsController {/api/seasons}: +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/seasons, GET} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/seasons/current, GET} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/seasons/:id, GET} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/seasons/:id/rules, GET} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RoutesResolver] SlackController {/api/slack}: +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/slack/seasons, POST} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/slack/season, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/slack/team, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/slack/team, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/slack/interact, POST} route +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RoutesResolver] TeamsController {/api/teams}: +0ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [RouterExplorer] Mapped {/api/teams, GET} route +1ms  
[0] [Nest] 3476 - 02/15/2024, 10:39:59 PM LOG [NestApplication] Nest application successfully started +50ms
```

8. Untuk melihat tampilan dari permainan ini silahkan kunjungi <http://localhost:8082/>

b. Langkah menjalankan Bot Starter Pack:

1. Ekstrak zip *Bot Starter Pack*, lalu buka folder hasil ekstraksinya dan buka terminal
2. Masuk ke root directory dari project

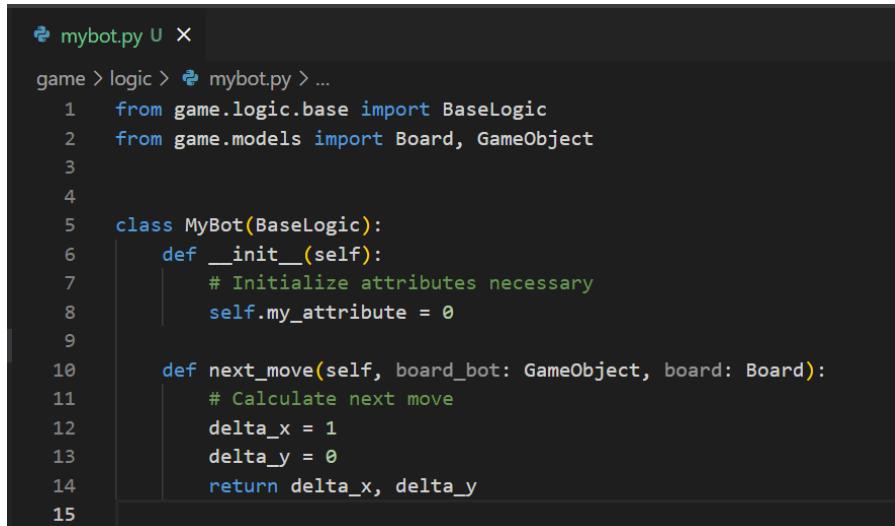
```
cd tubes1-IF2110-bot-starter-pack-1.0.1
```

3. Install dependencies menggunakan pip

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

4. Buat bot baru dengan suatu algoritma:

- Buat file baru pada direktori ./game/logic (misalkan mybot.py)
- Buat sebuah class yang merupakan turunan dari class BaseLogic
- Implementasikan constructor
- Implementasikan fungsi next_move pada class tersebut. Fungsi ini akan berisi algoritma greedy yang akan dibuat. Fungsi ini akan mengembalikan sebuah tuple (x, y) yang menunjukkan kemana bot akan bergerak. Fungsi ini akan dipanggil setiap satu detik sekali (bergantung aturan yang telah diatur) untuk menggerakan bot dalam permainan.



```
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
```

- Setelah algoritma selesai dibuat import class tersebut pada main.py dan daftarkan pada dictionary CONTROLLERS

```
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} ↓
17
```

5. Jalankan bot yang telah dibuat dengan perintah sebagai berikut

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

Untuk menjalankan banyak bot secara bersamaan bisa dengan menggunakan .bat atau .sh script kemudian menjalankannya dengan cara berikut

- Untuk windows

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

- Untuk Linux / (possibly) macOS

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

BAB 3

APLIKASI STRATEGI GREEDY

3.1 Pemilihan Fungsi Kelayakan

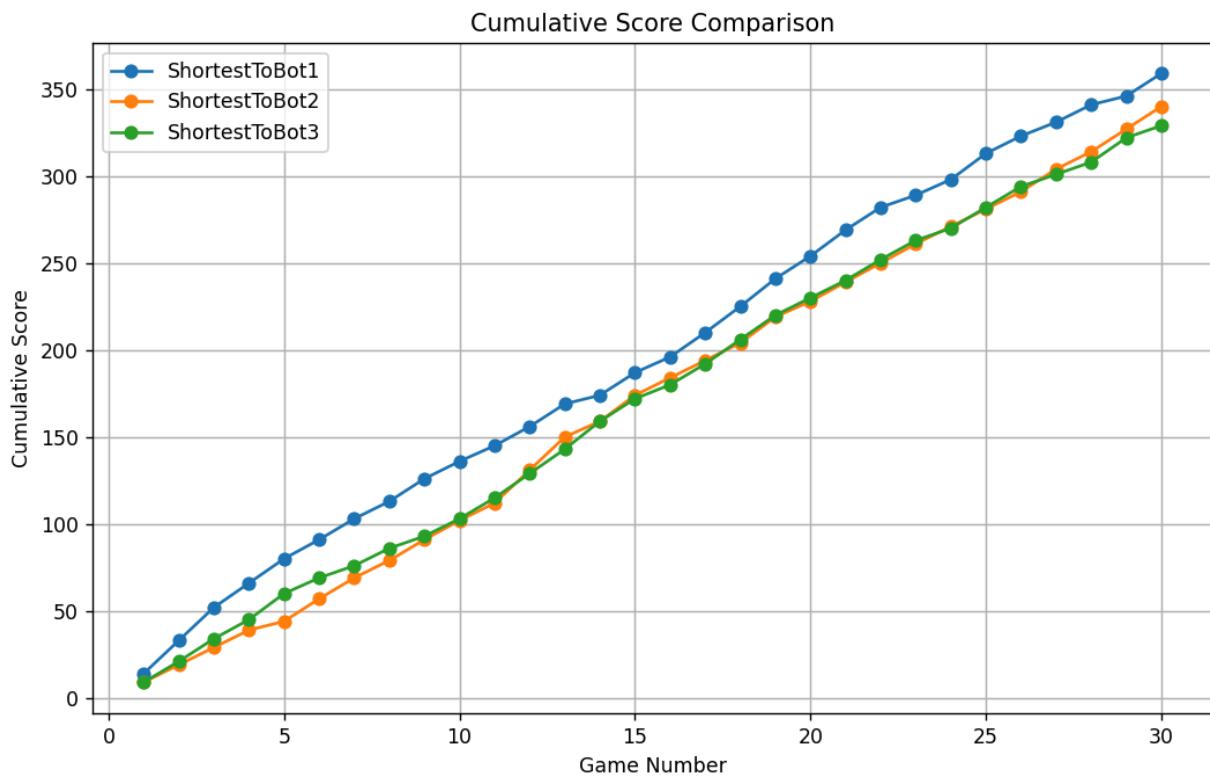
Salah satu komponen penting dalam algoritma greedy adalah fungsi kelayakan. Fungsi kelayakan sendiri merupakan fungsi yang menentukan apakah suatu kandidat solusi dalam sebuah masalah telah memenuhi semua kriteria yang telah ditetapkan untuk mencapai suatu objektif tertentu. Dalam tugas ini pemilihan fungsi kelayakan yang tepat sangat diperlukan dikarenakan fungsi kelayakan dapat membantu bot yang dibuat bergerak ke tujuan yang paling tepat untuk dapat memenangkan permainan. Dalam kasus permainan ini solusi yang dihasilkan oleh algoritma greedy adalah arah gerak dari bot menuju tujuan yang diinginkan. Oleh sebab itu fungsi kelayakan disini akan memilih mana tujuan yang paling tepat berdasarkan keadaan saat itu untuk mencapai kemenangan. Tujuan dari bot disini bisa berupa diamond, base, teleporter ataupun red button.

Untuk pemilihan fungsi kelayakan yang pertama kami akan melakukan tes kepada 3 bot dimana ketiga bot ini mempunyai algoritma greedy utama yang sama yaitu mencari diamond terdekat dari bot namun masing-masing dari bot tersebut memiliki fungsi kelayakan yang berbeda-beda. Fungsi kelayakan ini digunakan untuk menentukan kapan bot akan kembali ke base. Dalam pemilihan ini terdapat 3 kandidat dengan penjelasan sebagai berikut:

1. ShortestToBot1: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
2. ShortestToBot2: Bot hanya akan kembali ke base ketika inventory sudah penuh yaitu membawa lima diamond.
3. ShortestToBot3: Pada 30 detik pertama bot hanya akan kembali ke base ketika inventory sudah penuh sedangkan ketika waktu sudah tersisa 30 detik bot akan kembali ke base

ketika inventory sudah penuh setengah yaitu ketika inventory sudah membawa lebih dari sama dengan 3 diamond.

Kami melakukan pengetesan dengan menjalankan masing-masing bot secara individu dengan pengulangan sebanyak 30 kali secara berturut-turut. Melalui pengetesan tersebut didapatkan grafik sebagai berikut



Berdasarkan data grafik di atas dapat disimpulkan bahwa fungsi seleksi terbaik adalah fungsi seleksi yang digunakan pada ShortestBot1. Oleh sebab itu fungsi kelayakan tersebut akan digunakan pada keseluruhan bot lainnya dalam pengetesan-pengetesan berikutnya.

Fungsi kelayakan yang berikutnya adalah fungsi kelayakan yang dibuat agar bot akan kembali ke base ketika waktu yang tersisa tinggal sedikit. Kami tidak melakukan pengetesan untuk fungsi kelayakan ini dan langsung menggunakan ke dalam semua bot yang kami buat. Fungsi kelayakan ini kami gunakan karena jelas memberikan keuntungan dimana semua diamond yang telah dikumpulkan oleh bot kami tidak akan terbuang sia-sia karena akan selalu diusahakan untuk di bawa ke kembali base sebelum waktu habis.

Selain fungsi kelayakan di atas terdapat juga fungsi kelayakan lain yang kami pertimbangkan yaitu fungsi kelayakan untuk melakukan penyerangan terhadap bot lawan di sekitar. Namun fungsi kelayakan ini tidak kami pilih karena justru merugikan bot kami sendiri. Membuat algoritma penyerangan cukuplah kompleks dan sulit dan algoritma penyerangan yang kami buat cenderung selalu mengikuti bot lawan sehingga membuat waktu untuk mengumpulkan diamond terbuang sia-sia.

3.2 Alternatif Algoritma Greedy

3.2.1 Alternatif Greedy by Shortest to Bot

Algoritma greedy shortest to bot merupakan strategi algoritma dimana bot akan mengutamakan untuk mengambil diamond yang dekat dari dirinya terlebih dahulu. Jarak disini tidak mempertimbangkan penggunaan teleporter. Teleporter baru akan digunakan ketika tujuan dari bot sudah ditentukan. Ketika tujuan dari bot sudah ditentukan teleporter baru akan digunakan untuk memperpendek jarak yang akan ditempuh oleh bot jika memungkinkan. Bot tidak akan peduli apakah diamond yang akan dia ambil adalah diamond merah ataupun biru. Asalkan jarak dari diamond itu dengan dirinya merupakan yang terdekat maka bot akan langsung bergerak ke tempat itu dan mengambilnya.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki jarak terdekat dari bot.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target

bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.

- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Dalam algoritma ini pada intinya akan terjadi pengulangan sebanyak n kali dengan n adalah banyak diamond yang terdapat pada board. Pengulangan ini dilakukan untuk menghitung jarak dari bot pada saat itu menuju semua diamond yang ada di board. Setelah semua data jarak terkumpul semua, data tersebut akan diurutkan untuk mempermudah memperoleh mana diamond yang memiliki jarak terdekat dengan bot pada saat itu. Pengurutan ini dilakukan dengan menggunakan fungsi sorted bawaan dari python secara ascending yang mempunyai kompleksitas $O(n \log(n))$. Setelah diurutkan akan diambil data posisi diamond yang berada di paling depan dan kemudian bot akan bergerak ke posisi diamond tersebut. Pengulangan yang pertama yang dilakukan sebanyak n banyak diamond yang berada pada board memiliki kompleksitas waktu sebesar $O(n)$ sehingga kompleksitas waktu total untuk algoritma ini adalah $O(n \log(n))$. Perhitungan kompleksitas ini dilakukan hanya terhadap algoritma utamanya saja tidak termasuk kompleksitas untuk menjalankan fungsi-fungsi kelayakannya. Hal tersebut dikarenakan semua bot yang kami buat memiliki fungsi kelayakan yang sama oleh sebab itu kompleksitas waktunya pasti sama dan tidak perlu dipertimbangkan.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang memiliki jarak terdekat dengan bot pada suatu waktu tertentu. Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot berada pada daerah dengan konsentrasi diamond yang tinggi dengan jarak antar diamond yang berdekatan. Dengan keadaan yang seperti ini bot akan dengan mudah dan cepat untuk mengumpulkan diamond dan akan dengan cepat pula segera kembali ke base,
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukannya juga sangat kecil.

- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Jumlah diamond di board masih sangat banyak sehingga peluang jarak antar diamond berdekatan akan semakin tinggi.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Bot berada di daerah yang jarak antar diamondnya berjauhan
- Jumlah diamond di board tinggal sedikit sehingga peluang jarak antar diamond berjauhan akan sangat besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan. Keberadaan teleporter yang menghalangi seperti ini membuat perhitungan jarak terpendek yang telah dilakukan sebelumnya menjadi kurang valid.

3.2.2 Alternatif Greedy by Shortest to Base

Algoritma greedy shortest to base adalah algoritma greedy yang akan mencari diamond dengan jarak yang paling dekat dengan base. Sama seperti sebelumnya bot disini tidak akan memperdulikan apakah diamond tersebut adalah diamond merah atau diamond biru. Semakin dekat suatu diamond dengan base maka diamond tersebut akan semakin cepat diambil juga. Algoritma ini juga tidak berbeda dari algoritma sebelumnya dimana jarak diamond ke base tidak mempertimbangkan teleporter dan teleporter baru akan dipertimbangkan ketika tujuan dari bot sudah ditentukan.

- a. Pemetaan Elemen Greedy
- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
 - Himpunan Solusi: Himpunan diamond yang terpilih.

- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki jarak terdekat dari base baik dengan menggunakan teleporter maupun tidak.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Inti dari algoritma yang dibahas adalah proses iteratif yang berlangsung sebanyak n kali, dengan n merepresentasikan jumlah diamond di papan. Setiap iterasi bertujuan untuk menentukan jarak dari base ke tiap diamond. Informasi jarak ini kemudian dikumpulkan dan diurutkan dari yang terdekat hingga yang terjauh menggunakan fungsi sorted yang ada di Python, yang beroperasi dengan kompleksitas $O(n \log n)$. Urutan ini kemudian memudahkan pemilihan diamond terdekat, yang mana posisinya akan menjadi tujuan pergerakan bot. Proses iterasi untuk mengumpulkan jarak memiliki kompleksitas $O(n)$, menjadikan total kompleksitas keseluruhan algoritma menjadi $O(n \log n)$. Perhitungan kompleksitas ini hanya mengacu pada bagian kunci dari algoritma greedy dan tidak memasukkan kompleksitas dari fungsi kelayakan yang digunakan, mengingat semua bot dianggap memiliki fungsi seleksi yang seragam sehingga kompleksitasnya konsisten dan tidak dianggap variabel.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang memiliki jarak terdekat dengan base pada suatu waktu tertentu. Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Terdapat banyak diamond yang muncul di dekat base sehingga waktu pengumpulan dan pengembalian bot ke base akan sangat cepat.
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukan juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Waktu permainan tinggal sedikit dan banyak diamond di sekitaran base maka bot ini akan dapat mengamankan poin diamond di sekitaran base dengan sangat cepat.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Semua diamond yang tersisa di board memiliki jarak yang sangat jauh dari base.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.
- Semua diamond di board tiba-tiba ke-reset karena *red button* dan diamond yang baru muncul berada di sekitaran base sehingga membuat perhitungan jarak sebelumnya berubah dan membuat bot cenderung kembali ke sekitaran base dan mengabaikan diamond-diamond yang berada di sekitarnya yang sebelumnya ditargetkan. Kasus-kasus seperti ini akan membuat banyak waktu terbuang untuk bot berpindah kembali ke dekat base apalagi ketika pada saat itu bot sedang berada dari posisi yang sangat jauh dari base.

3.2.3 Alternatif Greedy by Shortest To Bot Base

Algoritma greedy ini akan mencari diamond dengan total jarak terdekat. Total jarak yang dimaksud disini adalah jarak dari bot ke diamond ditambah jarak dari diamond ke base. Sama seperti algoritma yang lain jarak disini tidak mempertimbangkan teleporter dan teleporter baru akan dipertimbangkan ketika tujuan dari bot tersebut sudah ditentukan. Diamond disini juga tidak mempedulikan apakah diamond tersebut adalah diamond biru atau diamond merah.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki total jarak terdekat dimana total jarak yang dimaksud adalah jarak dari bot ke diamond ditambah jarak dari diamond ke base.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Algoritma yang diuraikan bekerja dengan melakukan iterasi sejumlah n kali, di mana n adalah total diamond yang ada di board. Dalam setiap iterasi, algoritma menghitung total jarak dengan menambahkan jarak dari bot ke setiap diamond dengan jarak dari diamond tersebut ke base. Setelah mengumpulkan data total jarak ini, data tersebut diatur dalam urutan menaik berdasarkan kedekatannya menggunakan fungsi sorted Python, yang memiliki kompleksitas $O(n \log n)$. Berdasarkan urutan yang dihasilkan, algoritma memilih diamond dengan total jarak terpendek untuk dituju oleh bot. Proses penghitungan total jarak ini sendiri memiliki kompleksitas $O(n)$, sehingga kompleksitas keseluruhan algoritma menjadi $O(n \log n)$. Kompleksitas ini hanya mempertimbangkan inti dari algoritma greedy dan tidak termasuk kompleksitas dari fungsi kelayakan yang mungkin digunakan, karena asumsi bahwa semua bot di sini menggunakan fungsi kelayakan yang sama, membuat kompleksitasnya seragam dan bukan faktor yang berpengaruh.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang memiliki total jarak (bot ke diamond + diamond ke base) terdekat pada suatu waktu tertentu. Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukannya juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Bot akan mengambil diamond terakhir sebelum pulang karena pada kondisi ini algoritma ini akan langsung menghitung total jarak yang harus ditempuh bot hingga sampai ke base sehingga jarak terdekat yang diperoleh memang benar-benar jarak terdekat yang paling efisien.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.
- Jarak antara bot ke diamond dan diamond ke base terlalu tersebar sehingga total jarak yang dihitung menjadi terlalu acak dan membuat pergerakan bot sulit untuk diprediksi
- Semua diamond di board tiba-tiba ke-*reset* karena *red button* dan diamond yang baru muncul berada di sekitaran base sehingga membuat total jarak perhitungan berubah dan membuat bot cenderung kembali ke sekitaran base dan mengabaikan diamond-diamond yang berada di sekitarnya. Kasus-kasus seperti ini akan membuat banyak waktu terbuang untuk bot berpindah-pindah apalagi ketika pada saat itu bot sedang barada dari posisi yang sangat jauh dari base.

3.2.4 Alternatif Greedy by Shortest to Highest Reward

Algoritma greedy by Shortest to Highest Reward adalah algoritma greedy yang mengutamakan pengambilan diamond merah dengan point 2 terlebih dahulu baru diamond-diamond lainnya. Sama seperti algoritma shortest lainnya algoritma ini juga akan menghitung jarak dari bot ke semua diamond yang ada di board hanya saja nanti akan diurutkan berdasarkan point diamond terbesar dan jarak terdekat. Selama diamond merah masih ada di board maka bot akan selalu berusaha untuk mengambil diamond merah tersebut terlebih dahulu baru diamond biru atau diamond red button.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah, diamond biru, dan diamond red button.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki point tertinggi dan jarak terdekat dari bot.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Dalam algoritma ini pada intinya akan terjadi pengulangan sebanyak n kali dengan n adalah banyak diamond yang terdapat pada board. Pengulangan ini dilakukan untuk menghitung jarak dari bot pada saat itu menuju semua diamond yang ada di board. Setelah semua data jarak terkumpul semua, data tersebut akan diurutkan secara descending berdasarkan point diamond dan akan diurutkan secara ascending berdasarkan jarak dari diamond tersebut ke bot. Pengurutan ini dilakukan dengan menggunakan fungsi sorted bawaan dari Python yang mempunyai

kompleksitas $O(n \log(n))$. Setelah diurutkan akan diambil data posisi diamond yang berada di paling depan dan kemudian bot akan bergerak ke posisi diamond tersebut. Pengulangan untuk menghitung jarak yang dilakukan sebanyak n banyak diamond memiliki kompleksitas waktu sebesar $O(n)$ sehingga kompleksitas waktu total untuk algoritma ini adalah $O(n \log(n))$. Perhitungan kompleksitas ini dilakukan hanya terhadap algoritma utamanya saja tidak termasuk kompleksitas untuk menjalankan fungsi-fungsi kelayakannya. Hal tersebut dikarenakan semua bot yang kami buat memiliki fungsi kelayakan yang sama oleh sebab itu kompleksitas waktunya pasti sama dan tidak perlu dipertimbangkan.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan untuk mengambil diamond merah dengan jarak terdekat lebih dahulu baru diamond-diamond lainnya. Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukannya juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Bot berada di daerah dengan konsentrasi diamond merah yang tinggi dengan posisi yang berdekatan.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.
- Bot berada di daerah dengan konsentrasi diamond merah yang rendah dan jaraknya berjauhan sehingga bot menghabiskan terlalu banyak waktu di dalam perjalanan.

- Diamond merah di board sudah habis sehingga algoritma ini menjadi algoritma shortest to bot biasa yaitu mengumpulkan diamond biru terdekat.

3.2.5 Alternatif Greedy by Shortest to Highest Block

Algoritma greedy by Shortest to Highest Block adalah algoritma yang mencari block berukuran 3x3 dengan nilai terbesar dengan jarak terdekat dari bot. Nilai disini dihitung dengan menjumlahkan poin dari suatu diamond utama dengan semua diamond lain di sekitarnya dalam radius 1. Diamond utama adalah diamond yang berada di pusat block. Jarak disini adalah jarak dari diamond utama ke bot dan jarak disini sama seperti teleporter tidak dipertimbangkan. Teleporter baru akan dipertimbangkan ketika tujuan dari bot sudah ditemukan Algoritma.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki nilai tertinggi berdasarkan perhitungan poin diamond ditambah dengan poin semua diamond di sekitarnya dalam radius 1 dengan jarak dari diamond utama ke bot yang paling dekat.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Algoritma Shortest to Highest Block Reward beroperasi dengan memeriksa setiap blok diamond di papan permainan. Untuk setiap blok, algoritma menghitung nilai reward dengan cara berikut:

- Penghitungan Nilai Reward: Nilai reward sebuah blok dihitung dengan menjumlahkan nilai dari diamond utama di blok tersebut dan nilai dari semua diamond yang berada dalam radius 1 dari diamond utama. Proses ini melibatkan pemeriksaan hingga 8 posisi di sekitar diamond utama untuk menentukan apakah ada diamond lain yang terletak pada posisi tersebut.

Proses ini diulangi untuk setiap blok diamond yang tersedia, menghasilkan sebuah himpunan nilai reward untuk setiap blok. Kemudian, algoritma mengurutkan blok-blok tersebut berdasarkan nilai reward yang dihitung dari tertinggi ke terendah. Urutan ini dilakukan menggunakan fungsi sorted Python, yang memiliki kompleksitas $O(n \log n)$, di mana n adalah jumlah total blok diamond. Bot kemudian akan memilih blok dengan nilai reward tertinggi sebagai target untuk diambil.

Untuk setiap diamond, algoritma melakukan pemeriksaan hingga 8 posisi di sekitarnya, sehingga kompleksitas untuk langkah penghitungan nilai reward ini adalah $O(8n)$ atau $O(n)$ untuk setiap diamond. Namun, karena proses ini dilakukan untuk setiap diamond yang ada di board, kompleksitas total untuk seluruh papan permainan menjadi $O(n^2)$.

Kompleksitas keseluruhan algoritma ini, termasuk proses pengurutan, adalah $O(n^2 + n \log n)$. Dalam konteks ini, kompleksitas $O(n^2)$ mendominasi, sehingga kompleksitas keseluruhan dapat dianggap sebagai $O(n^2)$. Kompleksitas ini tidak termasuk kompleksitas dari fungsi kelayakan atau strategi tambahan yang mungkin digunakan oleh bot.

c. Analisis Efektifitas

Inti dari algoritma ini adalah untuk memprioritaskan blok diamond yang dikelilingi oleh banyak diamond lainnya, dengan jarak terdekat dari bot pada suatu waktu tertentu. Algoritma ini menjadi efektif ketika:

- Ada banyak kumpulan diamond yang berdekatan, sehingga bot dapat langsung menuju ke area tersebut tanpa mengutamakan diamond yang terisolasi. Dengan demikian, penggunaan langkah dan waktu menjadi lebih efisien.

- Terdapat banyak diamond yang muncul di dekat base sehingga waktu pengumpulan dan pengembalian bot ke base akan sangat cepat.
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukan juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Waktu permainan tinggal sedikit dan banyak diamond di sekitaran base maka bot ini akan dapat mengamankan poin diamond di sekitaran base dengan sangat cepat.

Namun, algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Semua diamond yang tersisa di board memiliki jarak yang sangat jauh dari base.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.
- Diamond di block diamond yang dituju sudah diambil beberapa oleh diri sendiri ataupun bot lain. Hal ini akan membuat perhitungan nilai block diamond berubah dan bot cenderung memiliki block diamond yang jaraknya jauh dibandingkan diamond yang jaraknya dekat. Akibat dari kelemahan ini adalah block diamond perlu dibagi dengan jaraknya ke bot sehingga bisa mendapatkan nilai perbandingan yang lebih baik lagi.

3.2.6 Alternatif Greedy by Highest Block Reward per Distance from Bot

Algoritma greedy by Highest Block Reward per Distance from Bot adalah strategi yang mencari blok dengan nilai tertinggi. Nilai ini dihitung dengan menjumlahkan nilai reward dari diamond utama dan semua diamond di sekitarnya dalam radius 1 (3x3 blok), kemudian dibagi dengan jarak antara diamond utama dan bot. Semakin dekat diamond utama ke bot, semakin tinggi nilainya. Algoritma ini tidak mempertimbangkan jarak antara diamond utama ke base dan

tidak menghitung jarak melalui teleporter. Teleporter hanya dipertimbangkan setelah tujuan bot telah ditentukan.

d. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki nilai tertinggi berdasarkan perhitungan reward diamond ditambah dengan reward semua diamond di sekitarnya dalam radius 1, dibagi dengan jarak dari bot ke diamond utama. Diamond dengan nilai tertinggi akan dipilih sebagai target selanjutnya.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

e. Analisis Efisiensi

Algoritma Highest Block Reward per Distance from Bot beroperasi dengan memeriksa setiap blok diamond di papan permainan. Untuk setiap blok, algoritma menghitung nilai reward dengan cara berikut:

- Penghitungan Nilai Reward: Nilai reward sebuah blok dihitung dengan menjumlahkan nilai dari diamond utama di blok tersebut dan nilai dari semua diamond yang berada dalam radius 1 dari diamond utama. Proses ini melibatkan pemeriksaan hingga 8 posisi di sekitar diamond utama untuk menentukan apakah ada diamond lain yang terletak pada posisi tersebut.

- Normalisasi dengan Jarak: Nilai reward yang dihitung kemudian dibagi dengan jarak dari bot ke blok diamond tersebut untuk mendapatkan nilai reward per unit jarak.

Proses ini diulangi untuk setiap blok diamond yang tersedia, menghasilkan sebuah himpunan nilai reward per unit jarak untuk setiap blok. Kemudian, algoritma mengurutkan blok-blok tersebut berdasarkan nilai reward per unit jarak yang dihitung dari tertinggi ke terendah. Urutan ini dilakukan menggunakan fungsi sorted Python, yang memiliki kompleksitas $O(n \log n)$, di mana n adalah jumlah total blok diamond.

Bot kemudian akan memilih blok dengan nilai reward per unit jarak tertinggi sebagai target untuk diambil. Dengan demikian, algoritma memprioritaskan blok-blok yang memberikan reward tinggi dengan jarak yang lebih dekat ke bot.

Untuk setiap diamond, algoritma melakukan pemeriksaan hingga 8 posisi di sekitarnya, sehingga kompleksitas untuk langkah penghitungan nilai reward ini adalah $O(8n)$ atau $O(n)$ untuk setiap diamond. Namun, karena proses ini dilakukan untuk setiap diamond, kompleksitas total untuk seluruh papan permainan menjadi $O(n^2)$.

Kompleksitas keseluruhan algoritma ini, termasuk proses pengurutan, adalah $O(n^2 + n \log n)$. Dalam konteks ini, kompleksitas $O(n^2)$ mendominasi, sehingga kompleksitas keseluruhan dapat dianggap sebagai $O(n^2)$. Kompleksitas ini tidak termasuk kompleksitas dari fungsi kelayakan atau strategi tambahan yang mungkin digunakan oleh bot.

f. Analisis Efektifitas

Inti dari algoritma ini adalah untuk memprioritaskan blok diamond yang dikelilingi oleh banyak diamond lainnya, dengan jarak terdekat dari bot pada suatu waktu tertentu. Algoritma ini menjadi efektif ketika:

- Ada kumpulan diamond yang berdekatan, sehingga bot dapat langsung menuju ke area tersebut tanpa mengutamakan diamond yang terisolasi. Dengan demikian, penggunaan langkah dan waktu menjadi lebih efisien.
- Terdapat banyak diamond yang muncul di dekat base sehingga waktu pengumpulan dan pengembalian bot ke base akan sangat cepat.
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tumbukan juga sangat kecil.

- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Waktu permainan tinggal sedikit dan banyak diamond di sekitaran base maka bot ini akan dapat mengamankan poin diamond di sekitaran base dengan sangat cepat.

Namun, algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Semua diamond yang tersisa di board memiliki jarak yang sangat jauh dari base.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.
- Anomali Jarak Pendek: Bot dapat terpikat oleh diamond yang sangat dekat meskipun nilai totalnya rendah, mengabaikan diamond dengan nilai lebih tinggi yang sedikit lebih jauh.

Selain itu, tanpa penimbangan yang optimal dalam menghitung nilai reward per unit jarak, bot mungkin terlalu memprioritaskan diamond yang sangat dekat atau terlalu fokus pada nilai reward tanpa mempertimbangkan jarak dengan cukup. Ini bisa menyebabkan bot mengambil keputusan yang kurang efisien, seperti terlalu cepat kembali ke base atau mengabaikan diamond dengan nilai tinggi yang sedikit lebih jauh.

3.2.7 Alternatif Greedy by Highest Block Reward per Distance from Base

Algoritma greedy Highest Block Reward per Distance from Base adalah strategi yang mencari blok dengan nilai tertinggi. Nilai ini dihitung dengan menjumlahkan nilai reward dari diamond utama dan semua diamond di sekitarnya dalam radius 1 (3x3 blok), kemudian dibagi dengan jarak antara diamond utama dan base. Semakin dekat diamond utama ke base, semakin tinggi nilainya. Algoritma ini tidak mempertimbangkan jarak antara diamond utama ke bot dan tidak menghitung jarak melalui teleporter. Teleporter hanya dipertimbangkan setelah tujuan bot telah ditentukan.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki nilai tertinggi berdasarkan perhitungan reward diamond ditambah dengan reward semua diamond di sekitarnya dalam radius 1, dibagi dengan jarak dari base ke diamond utama. Diamond dengan nilai tertinggi akan dipilih sebagai target selanjutnya.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Algoritma Highest Block Reward per Distance from Base beroperasi dengan memeriksa setiap blok diamond di papan permainan. Untuk setiap blok, algoritma menghitung nilai reward dengan cara berikut:

- Penghitungan Nilai Reward: Nilai reward sebuah blok dihitung dengan menjumlahkan nilai dari diamond utama di blok tersebut dan nilai dari semua diamond yang berada dalam radius 1 dari diamond utama. Proses ini melibatkan pemeriksaan hingga 8 posisi di sekitar diamond utama untuk menentukan apakah ada diamond lain yang terletak pada posisi tersebut.
- Normalisasi dengan Jarak: Nilai reward yang dihitung kemudian dibagi dengan jarak dari base ke blok diamond tersebut untuk mendapatkan nilai reward per unit jarak.

Proses ini diulangi untuk setiap blok diamond yang tersedia, menghasilkan sebuah himpunan nilai reward per unit jarak untuk setiap blok. Kemudian, algoritma mengurutkan blok-blok tersebut berdasarkan nilai reward per unit jarak yang dihitung dari tertinggi ke terendah. Urutan ini dilakukan menggunakan fungsi sorted Python, yang memiliki kompleksitas $O(n \log n)$, di mana n adalah jumlah total blok diamond.

Bot kemudian akan memilih blok dengan nilai reward per unit jarak tertinggi sebagai target untuk diambil. Dengan demikian, algoritma memprioritaskan blok-blok yang memberikan reward tinggi dengan jarak yang lebih dekat ke base.

Untuk setiap diamond, algoritma melakukan pemeriksaan hingga 8 posisi di sekitarnya, sehingga kompleksitas untuk langkah penghitungan nilai reward ini adalah $O(8n)$ atau $O(n)$ untuk setiap diamond. Namun, karena proses ini dilakukan untuk setiap diamond yang ada di board, kompleksitas total untuk seluruh papan permainan menjadi $O(n^2)$.

Kompleksitas keseluruhan algoritma ini, termasuk proses pengurutan, adalah $O(n^2 + n \log n)$. Dalam konteks ini, kompleksitas $O(n^2)$ mendominasi, sehingga kompleksitas keseluruhan dapat dianggap sebagai $O(n^2)$. Kompleksitas ini tidak termasuk kompleksitas dari fungsi kelayakan atau strategi tambahan yang mungkin digunakan oleh bot.

c. Analisis Efektifitas

Inti dari algoritma ini adalah untuk memprioritaskan blok diamond yang dikelilingi oleh banyak diamond lainnya, dengan jarak terdekat dari base pada suatu waktu tertentu. Algoritma ini menjadi efektif ketika:

- Ada kumpulan diamond yang berdekatan dekat dengan base, sehingga bot dapat langsung menuju ke area tersebut tanpa mengutamakan diamond yang terisolasi. Dengan demikian, penggunaan langkah dan waktu menjadi lebih efisien.
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukan juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Waktu permainan tinggal sedikit dan banyak diamond di sekitaran base maka bot ini akan dapat mengamankan poin diamond di sekitaran base dengan sangat cepat.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Semua diamond yang tersisa di board memiliki jarak yang sangat jauh dari base.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.

Dalam algoritma Greedy by Highest Block Reward per Distance from Base, terdapat beberapa anomali yang dapat mempengaruhi efektivitasnya:

- Anomali Diamond Terdekat ke Base:

Bot cenderung memilih diamond yang dekat dengan base, yang mungkin tidak selalu optimal. Misalnya, jika ada diamond dengan nilai lebih tinggi yang sedikit lebih jauh dari base, algoritma ini mungkin mengabaikannya karena fokus pada jarak ke base. Ini bisa mengakibatkan bot kehilangan kesempatan untuk mendapatkan poin lebih banyak karena terlalu fokus pada jarak daripada nilai reward.

- Pengabaian Posisi Bot:

Dalam situasi di mana bot berada jauh dari base, algoritma ini dapat mengabaikan diamond yang dekat dengan bot dan lebih menguntungkan untuk diambil terlebih dahulu. Sebagai contoh, jika bot berada di sudut yang berlawanan dari papan permainan, algoritma mungkin terus mendorong bot untuk bergerak menuju base meskipun ada diamond berharga di sekitar bot. Ini dapat menyebabkan penggunaan langkah yang tidak efisien dan memperlambat akumulasi poin.

3.2.8 Alternatif Greedy by Highest Block Reward per Distance from Bot and Base

Algoritma greedy Highest Block Reward per Distance from Bot and Base adalah strategi yang mencari blok dengan nilai tertinggi. Nilai ini dihitung dengan menjumlahkan nilai reward dari diamond utama dan semua diamond di sekitarnya dalam radius 1 (3x3 blok), kemudian dibagi dengan jumlah jarak antara diamond utama ke bot dan jarak dari diamond utama ke base.

Semakin dekat diamond utama ke bot dan base, semakin tinggi nilainya. Algoritma ini tidak menghitung jarak melalui teleporter. Teleporter hanya dipertimbangkan setelah tujuan bot telah ditentukan.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki nilai tertinggi berdasarkan perhitungan reward diamond ditambah dengan reward semua diamond di sekitarnya dalam radius 1, dibagi dengan jumlah jarak dari bot ke diamond utama dan jarak dari base ke diamond utama. Diamond dengan nilai tertinggi akan dipilih sebagai target selanjutnya.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Algoritma Highest Block Reward per Distance from Bot and Base beroperasi dengan memeriksa setiap blok diamond di papan permainan. Untuk setiap blok, algoritma menghitung nilai reward dengan cara berikut:

- Penghitungan Nilai Reward: Nilai reward sebuah blok dihitung dengan menjumlahkan nilai dari diamond utama di blok tersebut dan nilai dari semua diamond yang berada dalam radius 1 dari diamond utama. Proses ini melibatkan pemeriksaan hingga 8 posisi di

sekitar diamond utama untuk menentukan apakah ada diamond lain yang terletak pada posisi tersebut.

- Normalisasi dengan Jarak: Nilai reward yang dihitung kemudian dibagi dengan jumlah jarak dari base ke blok diamond dan jarak dari block diamond ke bot untuk mendapatkan nilai reward per unit jarak.

Proses ini diulangi untuk setiap blok diamond yang tersedia, menghasilkan sebuah himpunan nilai reward per unit jarak untuk setiap blok. Kemudian, algoritma mengurutkan blok-blok tersebut berdasarkan nilai reward per unit jarak yang dihitung dari tertinggi ke terendah. Urutan ini dilakukan menggunakan fungsi sorted Python, yang memiliki kompleksitas $O(n \log n)$, di mana n adalah jumlah total blok diamond.

Bot kemudian akan memilih blok dengan nilai reward per unit jarak tertinggi sebagai target untuk diambil. Dengan demikian, algoritma memprioritaskan blok-blok yang memberikan reward tinggi dengan jarak total (jarak bot ke block diamond + jarak block diamond ke base) yang lebih kecil.

Untuk setiap diamond, algoritma melakukan pemeriksaan hingga 8 posisi di sekitarnya, sehingga kompleksitas untuk langkah penghitungan nilai reward ini adalah $O(8n)$ atau $O(n)$ untuk setiap diamond. Namun, karena proses ini dilakukan untuk setiap diamond yang ada di board, kompleksitas total untuk seluruh papan permainan menjadi $O(n^2)$.

Kompleksitas keseluruhan algoritma ini, termasuk proses pengurutan, adalah $O(n^2 + n \log n)$. Dalam konteks ini, kompleksitas $O(n^2)$ mendominasi, sehingga kompleksitas keseluruhan dapat dianggap sebagai $O(n^2)$. Kompleksitas ini tidak termasuk kompleksitas dari fungsi kelayakan atau strategi tambahan yang mungkin digunakan oleh bot.

c. Analisis Efektifitas

Inti dari algoritma ini adalah untuk memprioritaskan blok diamond yang dikelilingi oleh banyak diamond lainnya, dengan jarak terdekat dari bot dan base pada suatu waktu tertentu. Algoritma ini menjadi efektif ketika:

- Ada kumpulan diamond yang berdekatan dekat dengan bot dan base, sehingga bot dapat langsung menuju ke area tersebut tanpa mengutamakan diamond yang terisolasi. Dengan demikian, penggunaan langkah dan waktu menjadi lebih efisien.

- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukan juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Waktu permainan tinggal sedikit dan banyak diamond di sekitaran base maka bot ini akan dapat mengamankan poin diamond di sekitaran base dengan sangat cepat.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Semua diamond yang tersisa di board memiliki jarak yang sangat jauh dari base.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan.

Selain itu, algoritma ini juga memiliki beberapa kekurangan lainnya:

- Dilema Pemilihan: Bot dapat mengalami kesulitan dalam memutuskan apakah akan mendekati diamond yang dekat dengan bot atau yang dekat dengan base, terutama jika nilai reward per unit jaraknya serupa.
- Kompleksitas Perhitungan: Perhitungan jarak ganda (dari bot ke diamond dan dari diamond ke base) dapat menyebabkan komputasi lebih banyak, terutama dalam papan permainan yang besar.
- Pengabaian Strategi Lawan: Seperti algoritma lainnya, algoritma ini juga kurang mempertimbangkan posisi dan strategi lawan, yang bisa menjadi kelemahan dalam situasi kompetitif.

3.2.9 Alternatif Greedy by Highest Reward Per Distance From Bot

Algoritma greedy highest reward per distance bot merupakan strategi algoritma dimana bot akan mengutamakan untuk mengambil diamond yang memiliki nilai poin dibagi jarak dari bot tertinggi. Jarak disini tidak mempertimbangkan penggunaan teleporter. Teleporter baru akan digunakan ketika tujuan dari bot sudah ditentukan. Ketika tujuan dari bot sudah ditentukan teleporter baru akan digunakan untuk memperpendek jarak yang akan ditempuh oleh bot jika memungkinkan. Bot memperdulikan efisiensi dari nilai poin yang diperoleh dengan jarak yang ditempuh.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki nilai dari poin dibagi dengan jarak diamond ke bot yang paling tinggi.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Dalam algoritma ini pada intinya akan terjadi pengulangan sebanyak n kali dengan n adalah banyak diamond yang terdapat pada board. Pengulangan ini dilakukan untuk menghitung nilai diamond dibagi jarak dari bot pada saat itu menuju semua diamond yang ada di board sehingga mendapatkan sekumpulan nilai. Setelah semua data terkumpul, data tersebut akan diurutkan

untuk mempermudah memperoleh mana diamond yang memiliki nilai perhitungan paling besar. Pengurutan ini dilakukan dengan menggunakan fungsi sorted bawaan dari python secara ascending yang mempunyai kompleksitas $O(n \log(n))$. Setelah diurutkan akan diambil data posisi diamond yang berada di paling depan dan kemudian bot akan bergerak ke posisi diamond tersebut. Pengulangan yang pertama yang dilakukan sebanyak n banyak diamond yang berada pada board memiliki kompleksitas waktu sebesar $O(n)$ sehingga kompleksitas waktu total untuk algoritma ini adalah $O(n \log(n))$. Perhitungan kompleksitas ini dilakukan hanya terhadap algoritma utamanya saja tidak termasuk kompleksitas untuk menjalankan fungsi-fungsi kelayakannya. Hal tersebut dikarenakan semua bot yang kami buat memiliki fungsi kelayakan yang sama oleh sebab itu kompleksitas waktunya pasti sama dan tidak perlu dipertimbangkan.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang nilai terbesar dari hasil bagi point diamond dengan jaraknya dari bot. Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot berada pada daerah dengan konsentrasi diamond yang tinggi dengan jarak antar diamond yang berdekatan. Dengan keadaan yang seperti ini bot akan dengan mudah dan cepat untuk mengumpulkan diamond dan akan dengan cepat pula segera kembali ke base,
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukan juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Jumlah diamond di board masih sangat banyak sehingga peluang jarak antar diamond berdekatan akan semakin tinggi.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Bot berada di daerah yang jarak antar diamondnya berjauhan

- Jumlah diamond di board tinggal sedikit sehingga peluang jarak antar diamond berjauhan akan sangat besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan. Keberadaan teleporter yang menghalangi seperti ini membuat perhitungan jarak terpendek yang telah dilakukan sebelumnya menjadi kurang valid.
- Jika terdapat nilai poin diamond dibagi jarak terhadap bot yang paling kecil sama namun berbeda jarak maka pergerakan bot tidak dapat diperkirakan akan mengambil yang memiliki jarak lebih dekat atau jauh.

3.2.10 Alternatif Greedy by Highest Reward Per Distance From Bot and Base

Algoritma greedy highest reward per distance bot merupakan strategi algoritma dimana bot akan mengutamakan untuk mengambil diamond yang memiliki nilai poin dibagi dengan jarak total dari bot ke diamond dan dari diamond ke base yang tertinggi. Jarak disini tidak mempertimbangkan penggunaan teleporter. Teleporter baru akan digunakan ketika tujuan dari bot sudah ditentukan. Ketika tujuan dari bot sudah ditentukan teleporter baru akan digunakan untuk memperpendek jarak yang akan ditempuh oleh bot jika memungkinkan. Bot memperdulikan efisiensi dari nilai poin yang diperoleh dengan jarak total yang ditempuh.

- a. Pemetaan Elemen Greedy
- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
 - Himpunan Solusi: Himpunan diamond yang terpilih.
 - Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
 - Fungsi Seleksi: Memilih diamond yang memiliki nilai dari poin dibagi dengan jarak total (bot ke diamond dan diamond ke base) yang paling tinggi.
 - Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau

bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.

- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Dalam algoritma ini pada intinya akan terjadi pengulangan sebanyak n kali dengan n adalah banyak diamond yang terdapat pada board. Pengulangan ini dilakukan untuk menghitung nilai diamond dibagi dengan jarak total (bot ke diamond dan diamond ke base) pada semua diamond yang ada di board sehingga mendapatkan sekumpulan nilai. Setelah semua data terkumpul, data tersebut akan diurutkan untuk mempermudah memperoleh mana diamond yang memiliki nilai perhitungan paling besar. Pengurutan ini dilakukan dengan menggunakan fungsi sorted bawaan dari python secara ascending yang mempunyai kompleksitas $O(n \log(n))$. Setelah diurutkan akan diambil data posisi diamond yang berada di paling depan dan kemudian bot akan bergerak ke posisi diamond tersebut. Pengulangan yang pertama yang dilakukan sebanyak n banyak diamond yang berada pada board memiliki kompleksitas waktu sebesar $O(n)$ sehingga kompleksitas waktu total untuk algoritma ini adalah $O(n \log(n))$. Perhitungan kompleksitas ini dilakukan hanya terhadap algoritma utamanya saja tidak termasuk kompleksitas untuk menjalankan fungsi-fungsi kelayakannya. Hal tersebut dikarenakan semua bot yang kami buat memiliki fungsi kelayakan yang sama oleh sebab itu kompleksitas waktunya pasti sama dan tidak perlu dipertimbangkan.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang nilai terbesar dari hasil bagi point diamond dengan jarak total (bot ke diamond dan diamond ke base) . Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot berada pada daerah dengan konsentrasi diamond yang tinggi dengan jarak antar diamond yang berdekatan. Dengan keadaan yang seperti ini bot akan dengan mudah dan cepat untuk mengumpulkan diamond dan akan dengan cepat pula segera kembali ke base,
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukannya juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Jumlah diamond di board masih sangat banyak sehingga peluang jarak antar diamond berdekatan akan semakin tinggi.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Bot berada di daerah yang jarak antar diamondnya berjauhan
- Jumlah diamond di board tinggal sedikit sehingga peluang jarak antar diamond berjauhan akan sangat besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan. Keberadaan teleporter yang menghalangi seperti ini membuat perhitungan jarak terpendek yang telah dilakukan sebelumnya menjadi kurang valid.

3.2.11 Alternatif Greedy by Highest Reward Per Distance From Base

Algoritma greedy highest reward per distance bot merupakan strategi algoritma dimana bot akan mengutamakan untuk mengambil diamond yang memiliki nilai tertinggi dari poin dibagi dengan jarak diamond ke base. Jarak disini tidak mempertimbangkan penggunaan teleporter. Teleporter baru akan digunakan ketika tujuan dari bot sudah ditentukan. Ketika tujuan dari bot sudah ditentukan teleporter baru akan digunakan untuk memperpendek jarak yang akan

ditempuh oleh bot jika memungkinkan. Bot memperdulikan efisiensi dari nilai poin yang diperoleh dengan jarak total yang ditempuh.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah dan diamond biru.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki nilai dari poin dibagi dengan jarak (diamond ke base) yang paling tinggi.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond. Selain itu, bot akan ke diamond red button jika target bot sedang tidak ke base dan jarak ke diamond red button lebih kecil dari jarak ke target diamond yang dipilih.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Dalam algoritma ini pada intinya akan terjadi pengulangan sebanyak n kali dengan n adalah banyak diamond yang terdapat pada board. Pengulangan ini dilakukan untuk menghitung nilai diamond dibagi dengan jarak (diamond ke base) pada semua diamond yang ada di board sehingga mendapatkan sekumpulan nilai. Setelah semua data terkumpul, data tersebut akan diurutkan untuk mempermudah memperoleh mana diamond yang memiliki nilai perhitungan paling besar. Pengurutan ini dilakukan dengan menggunakan fungsi sorted bawaan dari python secara ascending yang mempunyai kompleksitas $O(n \log(n))$. Setelah diurutkan akan diambil data posisi diamond yang berada di paling depan dan kemudian bot akan bergerak ke posisi diamond tersebut. Pengulangan yang pertama yang dilakukan sebanyak n banyak diamond yang berada pada board memiliki kompleksitas waktu sebesar $O(n)$ sehingga kompleksitas waktu total

untuk algoritma ini adalah $O(n \log(n))$. Perhitungan kompleksitas ini dilakukan hanya terhadap algoritma utamanya saja tidak termasuk kompleksitas untuk menjalankan fungsi-fungsi kelayakannya. Hal tersebut dikarenakan semua bot yang kami buat memiliki fungsi kelayakan yang sama oleh sebab itu kompleksitas waktunya pasti sama dan tidak perlu dipertimbangkan.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang nilai terbesar dari hasil bagi point diamond dengan jarak total (diamond ke base) . Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot berada pada daerah dengan konsentrasi diamond yang tinggi dengan jarak antar diamond yang berdekatan. Dengan keadaan yang seperti ini bot akan dengan mudah dan cepat untuk mengumpulkan diamond dan akan dengan cepat pula segera kembali ke base,
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukannya juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Jumlah diamond di board masih sangat banyak sehingga peluang jarak antar diamond berdekatan akan semakin tinggi.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Bot berada di daerah yang jarak antar diamondnya berjauhan
- Jumlah diamond di board tinggal sedikit sehingga peluang jarak antar diamond berjauhan akan sangat besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan. Keberadaan teleporter yang

menghalangi seperti ini membuat perhitungan jarak terpendek yang telah dilakukan sebelumnya menjadi kurang valid.

3.2.12 Alternatif Greedy by Shortest to Bot V2

Algoritma ini adalah modifikasi dari algoritma shortest to bot sebelumnya. Algoritma ini akan mencari diamond dengan jarak terdekat dari bot namun dengan mempertimbangkan teleporter juga. Algoritma ini juga tidak peduli jenis diamondnya apakah diamond itu merah, biru, atau red button. Pada algoritma ini ketika bot ingin mengambil diamond terakhir bot akan menggunakan algoritma shortest to bot base untuk meningkatkan efisiensi jarak yang akan ditempuh. Diamond yang terpilih disini juga akan di seleksi kembali apakah jarak dari diamond tersebut ke base lebih dari 8 langkah atau tidak. Jika memang tidak ada yang jaraknya kurang dari 8 langkah dari base maka baru akan dicari diamond yang terdekat sisanya.

a. Pemetaan Elemen Greedy

- Himpunan Kandidat: Himpunan diamond yang tersedia di dalam board permainan meliputi diamond merah, diamond biru, dan diamond red button.
- Himpunan Solusi: Himpunan diamond yang terpilih.
- Fungsi Solusi: Menjumlahkan seluruh poin diamond yang telah didapatkan.
- Fungsi Seleksi: Memilih diamond yang memiliki jarak terdekat dari bot baik dengan menggunakan teleporter maupun tidak.
- Fungsi kelayakan: Bot akan kembali ke base jika diamond di inventory sudah sama dengan 4 dan dan diamond terdekat yang bisa diambil adalah diamond merah dengan point dua atau bot akan kembali ke base ketika inventory penuh atau bot akan kembali ke base ketika inventory sudah berisi lebih dari sama dengan 3 dan jarak ke base lebih dekat dibandingkan jarak ke diamond terdekat berikutnya, bot juga akan kembali ke base jika waktu yang tersisa tidak cukup lagi untuk mengumpulkan diamond dan bot sudah membawa minimal satu diamond.
- Fungsi objektif: Jumlah poin diamond yang didapatkan maksimum.

b. Analisis Efisiensi

Dalam algoritma ini pada intinya akan terjadi pengulangan sebanyak n kali dengan n adalah banyak diamond yang terdapat pada board. Pengulangan ini dilakukan untuk menghitung jarak dari bot pada saat itu menuju semua diamond yang ada di board. Perhitungan ini juga akan memperhitungkan penggunaan teleporter. Karena pada kasus ini penggunaan teleporter hanya ada 1 pasang maka perhitungan teleporter tidak akan terlalu mengganggu kompleksitasnya. Setelah semua data jarak terkumpul semua, data tersebut akan diurutkan untuk mempermudah memperoleh mana diamond yang memiliki jarak terdekat dengan bot pada saat itu. Pengurutan ini dilakukan dengan menggunakan fungsi sorted bawaan dari python secara ascending yang mempunyai kompleksitas $O(n \log(n))$. Setelah diurutkan akan diambil data posisi diamond yang berada di paling depan dan kemudian bot akan bergerak ke posisi diamond tersebut. Pengulangan yang pertama yang dilakukan sebanyak n banyak diamond yang berada pada board memiliki kompleksitas waktu sebesar $O(n)$ sehingga kompleksitas waktu total untuk algoritma ini adalah $O(n \log(n))$. Perhitungan kompleksitas ini dilakukan hanya terhadap algoritma utamanya saja tidak termasuk kompleksitas untuk menjalankan fungsi-fungsi kelayakannya. Hal tersebut dikarenakan semua bot yang kami buat memiliki fungsi kelayakan yang sama oleh sebab itu kompleksitas waktunya pasti sama dan tidak perlu dipertimbangkan.

c. Analisis Efektifitas

Algoritma ini pada intinya akan mengutamakan diamond yang memiliki jarak terdekat dengan bot pada suatu waktu tertentu. Dengan algoritma yang seperti itu maka algoritma ini akan efektif ketika:

- Bot berada pada daerah dengan konsentrasi diamond yang tinggi dengan jarak antar diamond yang berdekatan. Dengan keadaan yang seperti ini bot akan dengan mudah dan cepat untuk mengumpulkan diamond dan akan dengan cepat pula segera kembali ke base,
- Bot tidak berada di posisi yang dekat dengan bot lawan sehingga peluang tabrakan antar bot akan semakin kecil dan peluang bot akan kehilangan diamond karena tubrukan juga sangat kecil.
- Terdapat teleporter yang posisinya dekat dengan bot yang mempercepat bot untuk pergi mengambil diamond ataupun untuk kembali ke base.
- Jumlah diamond di board masih sangat banyak sehingga peluang jarak antar diamond berdekatan akan semakin tinggi.

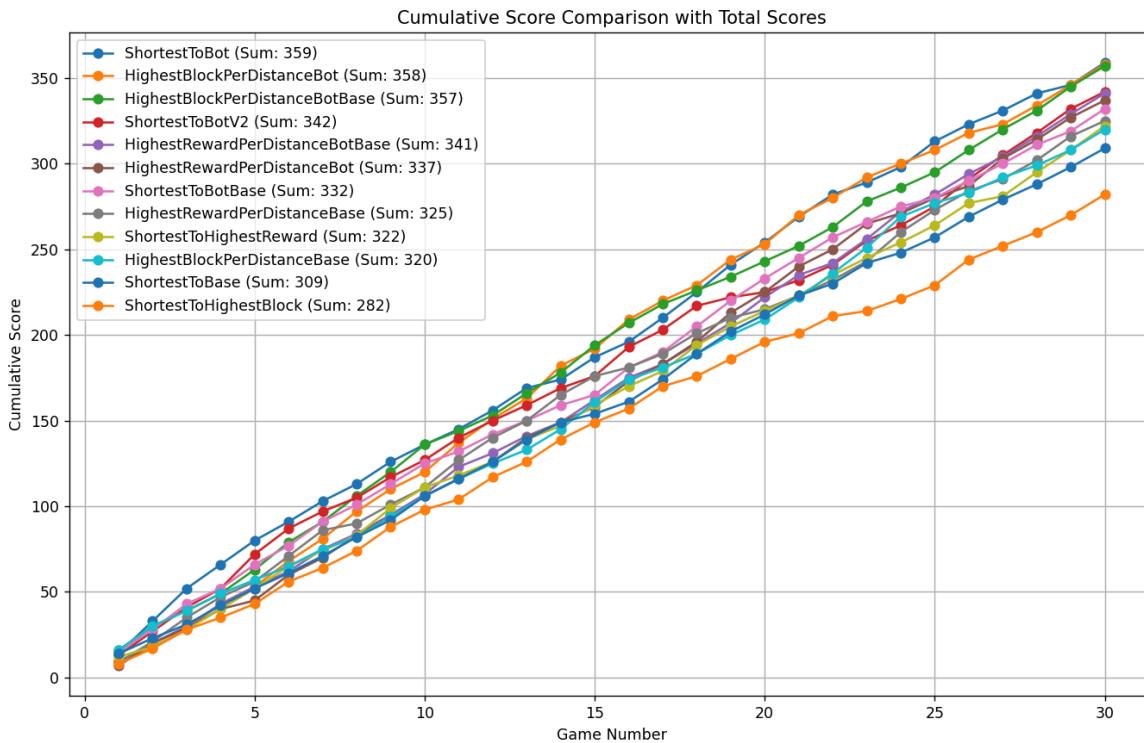
- Bot akan mengambil diamond terakhir sebelum pulang karena pada kondisi ini algoritma ini akan langsung menghitung total jarak yang harus ditempuh bot hingga sampai ke base sehingga jarak terdekat yang diperoleh memang benar-benar jarak terdekat yang paling efisien.

Dan algoritma ini akan kurang efektif ketika:

- Bot berada di posisi yang dekat dengan bot lawan sehingga peluang diamond hilang akibat tabrakan antar bot akan semakin besar.
- Bot berada di daerah yang jarak antar diamondnya berjauhan
- Jumlah diamond di board tinggal sedikit sehingga peluang jarak antar diamond berjauhan akan sangat besar.
- Teleporter tidak berada di posisi yang dapat membantu bot untuk memperpendek jarak tempuhnya ke suatu tujuan.
- Terdapat teleporter yang menghalangi gerak bot sehingga bot akan memerlukan beberapa gerakan tambahan untuk mencapai tujuan diinginkan. Keberadaan teleporter yang menghalangi seperti ini membuat perhitungan jarak terpendek yang telah dilakukan sebelumnya menjadi kurang valid.

3.3 Strategi Greedy yang Dipilih

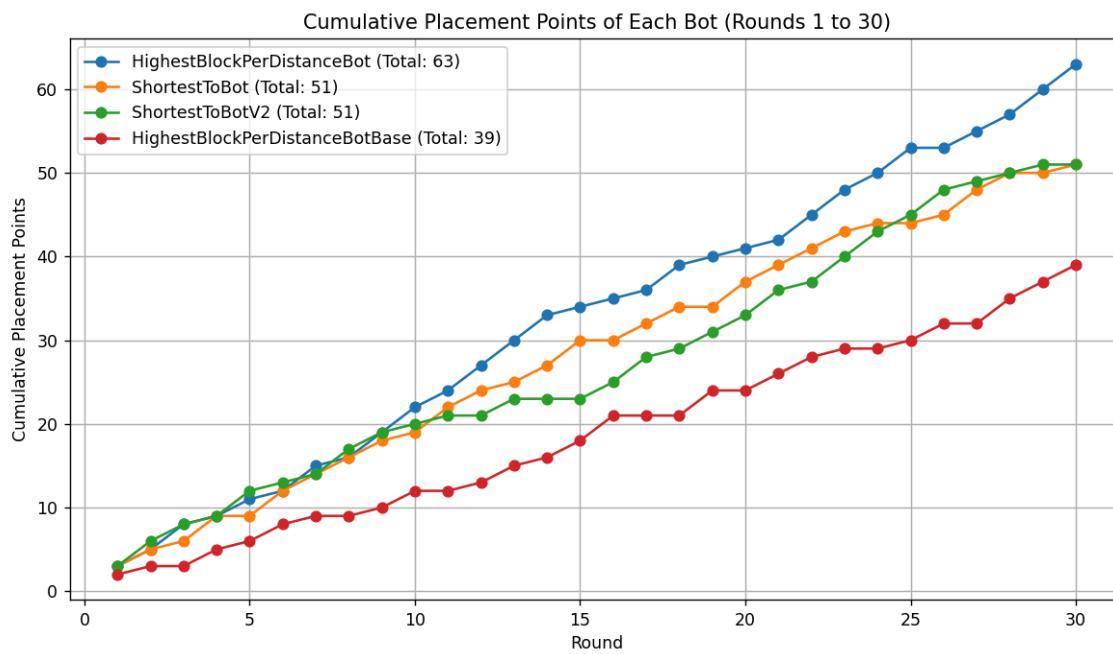
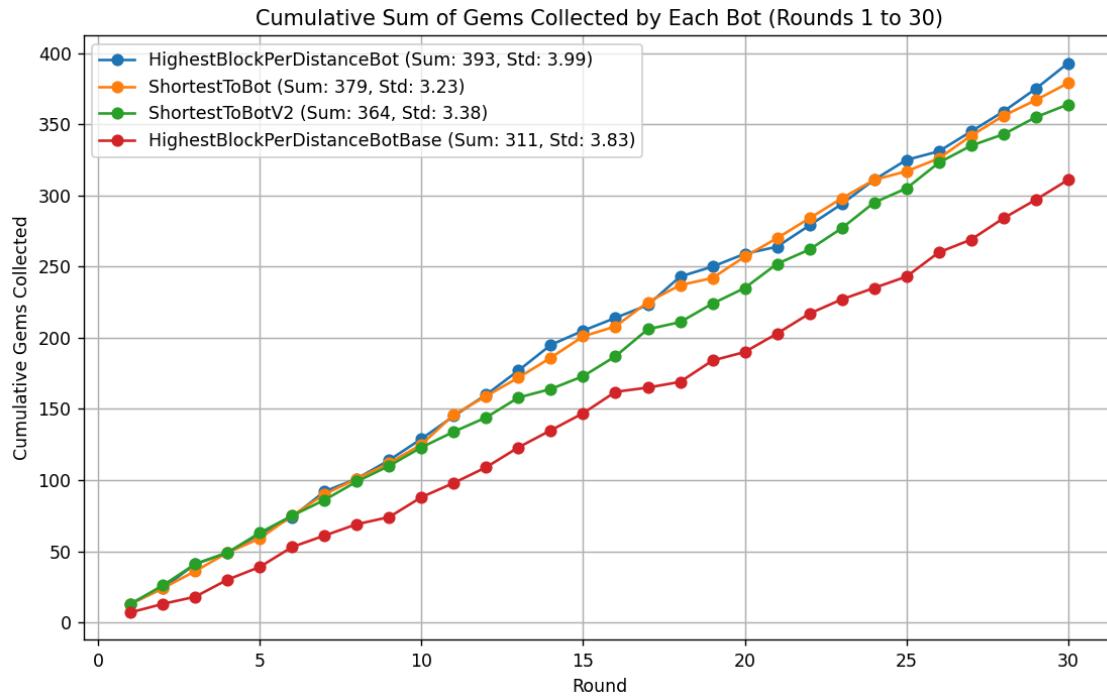
Dalam studi ini, 12 varian algoritma greedy telah diuji untuk mengevaluasi kinerjanya dalam permainan pengumpulan diamond. Setiap algoritma menjalani 30 pertandingan terpisah, dan penting untuk dicatat bahwa kondisi papan permainan tidak di-reset setelah setiap pertandingan. Ini berarti bahwa setiap pertandingan dimulai dari kondisi akhir pertandingan sebelumnya, memungkinkan algoritma untuk mengembangkan strategi yang konsisten sepanjang serangkaian pertandingan.



Dari semua algoritma yang diuji, empat algoritma yang paling berhasil dalam mengumpulkan jumlah diamond terbesar secara kumulatif adalah:

- Algoritma Greedy Shortest to Bot: Mengumpulkan total 359 diamond.
- Algoritma Greedy Highest Block Reward per Distance from Bot: Mengumpulkan total 358 diamond.
- Algoritma Greedy Highest Block Reward per Distance from Bot and Base: Mengumpulkan total 357 diamond.
- Algoritma Greedy Shortest to Bot V2: Mengumpulkan total 342 diamond.

Keempat algoritma teratas ini kemudian diuji dalam pertandingan *head-to-head* sebanyak 30 kali. Hasilnya, Algoritma Greedy by Highest Block Reward per Distance from Bot terbukti paling superior dengan total 393 diamond yang dikumpulkan. Berdasarkan sistem poin, di mana juara pertama mendapat 3 poin, juara kedua mendapat 2 poin, dan juara ketiga mendapat 1 poin, algoritma ini berhasil mengumpulkan 63 poin dan menempati posisi pertama sebanyak 14 kali dari 30 pertandingan.



Berdasarkan hasil pengujian ini, Algoritma Greedy Highest Block Reward per Distance from Bot dinyatakan sebagai algoritma terbaik dalam konteks permainan ini. Algoritma ini unggul dalam mengidentifikasi dan menargetkan kelompok diamond, memungkinkan bot untuk mengumpulkan banyak diamond dalam satu perjalanan. Keberhasilan algoritma ini terutama

didasarkan pada penyeimbangan yang cermat antara nilai reward dari diamond dan jarak dari bot ke diamond.

Dengan fokus pada kelompok diamond, bot tidak hanya meningkatkan jumlah diamond yang dikumpulkan tetapi juga memaksimalkan efisiensi penggunaan langkah dan waktu. Pertimbangan jarak dalam algoritma ini memastikan bahwa bot mengambil rute yang optimal, menghindari pemborosan langkah yang tidak perlu dan mempercepat proses pengumpulan. Hal ini sangat penting dalam situasi di mana waktu dan langkah terbatas.

Selain itu, algoritma ini menunjukkan adaptabilitas yang baik dalam berbagai situasi permainan, mampu merespons dengan cepat terhadap perubahan kondisi papan dan posisi diamond. Oleh karena itu, Algoritma Greedy Highest Block Reward per Distance from Bot tidak hanya berhasil dalam pengumpulan diamond secara kuantitatif tetapi juga dalam efektifitas, efisiensi, dan adaptabilitas, menjadikannya pilihan yang lebih unggul dibandingkan dengan algoritma lain seperti Shortest to Bot.

Berbagai macam pengujian ini kami lakukan untuk mendapatkan data asli dari sebuah bot permainan sehingga proses pemilihan bot terbaik akan lebih mudah ditentukan. Jika pemilihan bot hanya dilakukan berdasarkan analisis efisiensi dan efektifitas kami rasa itu masih sedikit sulit untuk dapat menemukan suatu bot yang paling unggul dibandingkan dengan bot-bot lainnya. Berdasarkan hasil analisis efisiensi yang telah kami lakukan kami juga menemukan bahwa semua algoritma yang kami buat cenderung mempunyai kompleksitas waktu yang cenderung tidak berbeda jauh sehingga analisis efisiensi tidak bisa dijadikan patokan utama dalam proses pemilihan bot terbaik. Kompleksitas waktu disini juga dipengaruhi oleh banyaknya objek pada board permainan namun dengan konfigurasi permainan default ini kompleksitas waktu tidak akan berpengaruh terlalu besar. Berdasarkan hasil analisis efektifitas juga, kami menemukan bahwa masing-masing bot memiliki keunggulan dan kelemahannya masing-masing bergantung pada situasi board permainan pada suatu waktu oleh sebab itu jika pemilihan bot dilakukan hanya berdasarkan dengan analisis efektifitas ini maka akan sulit untuk menentukan mana bot yang terbaik diantara bot-bot yang ada.

Semua pengujian yang telah kami lakukan disini dilakukan pada kondisi konfigurasi permainan yang default sesuai dengan aturan yang ada.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy yang Dipilih

Berikut merupakan implementasi dari algoritma greedy yang dipilih:

Pseudocode Algoritma Greedy by Highest Block Reward per Distance From Bot

```
Class HighestBlockPerDistanceBotLogic:
    Initialize:
        // Set arah gerakan bot
        Set directions to [(1, 0), (0, 1), (-1, 0), (0, -1)]
        // Inisialisasi posisi tujuan awal sebagai None
        Set goal_position to None
        // Inisialisasi arah gerakan saat ini
        Set current_direction to 0
        // Atur apakah bot akan menyerang bot lain
        Set attack_other_bots to False
        // Atur apakah bot akan mundur berdasarkan waktu yang tersisa
        Set time_based_retreat to True

        //fungsi untuk menghitung total poin diamond pada suatu block 3x3 (poin diamond utama
        dan
        //dan diamond disekitarnya pada radius 1)
        Function get_surroundings_points(diamond, list_of_diamonds):
            // Tentukan posisi sekitar diamond untuk diperiksa
            Set list_to_check to [(-1, 1), (0, 1), (1, 1), (-1, 0), (1, 0), (-1, -1), (0, -1),
            (1, -1)]
            // Mulai dengan nilai diamond itu sendiri
            Set points to diamond's value
            // Periksa setiap posisi di sekitar diamond
            For each delta in list_to_check:
                Calculate coordinate_x and coordinate_y
                // Tambahkan nilai diamond di sekitar ke total poin
                For each position, point in list_of_diamonds:
                    If position matches coordinates:
                        Add point to points
                // Kembalikan total poin dari diamond dan sekitarnya
            Return points
```

```

//fungsi untuk menghitung jarak antar 2 objek pada permainan
Function get_distance(current_position, target_position):
    // Hitung jarak Manhattan antara dua posisi
    Return absolute difference in x and y coordinates


    //fungsi untuk menghitung nilai dari suatu block (total poin diamond di bagi jarak
diamond
//utama ke bot)
Function calculate_benefit(point, distance):
    // Hitung manfaat dengan membagi nilai diamond dengan jarak
    If distance is 0:
        Return 0
    Else:
        Return (point / distance) * 100000


    //Fungsi untuk mengecek apakah jika menggunakan teleporter jarak bisa menjadi lebih
dekat
Function is_teleporting_closer(bot_position, target_position, teleporter_pairs):
    // Hitung jarak langsung antara bot dan tujuan
    Set direct_distance to get_distance(bot_position, target_position)
    // Periksa apakah menggunakan teleporter lebih dekat
    For each teleporter_pair in teleporter_pairs:
        Calculate distance_via_teleporter_1 and distance_via_teleporter_2
        // Jika ya, kembalikan True dan posisi teleporter
        If either is less than direct_distance:
            Return True, teleporter
        // Jika tidak, kembalikan False dan None
        Return False, None


    //Fungsi untuk menentukan arah gerak bot berdasarkan tujuan yang sudah ditentukan,
fungsi
//ini akan lebih mengutamakan pergerakan ke arah vertikal lebih dahulu
Function get_direction_yfirst(current_x, current_y, dest_x, dest_y):
    // Hitung perubahan posisi dengan memprioritaskan sumbu y
    Calculate delta_x and delta_y using clamp function
    If delta_y is not 0:
        Set delta_x to 0
    // Kembalikan arah gerakan
    Return (delta_x, delta_y)

```

```

//Fungsi untuk menghandle masalah ketika arah gerak tidak valid (0, 0) dan menyebabkan
//error dan bot diam di tempat, dengan fungsi ini setidaknya bot akan bergerak dan
akan

//memulai perhitungan ulang
Function handle_not_moving(current_position, board_width, board_height):
    // Tangani kasus ketika bot tidak bergerak
    Handle edge cases for when the bot is not moving
    // Kembalikan arah gerakan yang sesuai
    Return appropriate direction

//fungsi utama yang berisi algoritma greedy yang digunakan, fungsi ini akan dipanggil
//setiap beberapa saat untuk menggerakkan bot dalam board permainan
Function next_move(board_bot, board):
    // Dapatkan daftar objek permainan, diamond, dan teleporter
    Get list of game objects, diamonds, and teleporters
    // Hitung poin sekitar, jarak, dan manfaat untuk setiap diamond
    Calculate surrounding_points, distance, and benefit for each diamond
    // Urutkan diamond berdasarkan manfaat
    Sort diamonds based on benefit
    // Tentukan posisi tujuan berdasarkan properti bot dan posisi diamond
    Determine goal_position based on bot properties and diamond positions
    // Tangani tombol refresh (red button) dan mundur berdasarkan waktu
    Handle refresh buttons and time-based retreat
    // Periksa apakah teleportasi lebih dekat ke tujuan
    Check if teleporting is closer to the goal
    // Tentukan arah gerakan berdasarkan posisi tujuan
    Determine movement direction based on goal_position
    // Tangani kasus ketika bot tidak bergerak
    Handle not moving cases
    // Kembalikan arah gerakan (delta_x, delta_y)
    Return movement direction (delta_x, delta_y)

```

4.2 Struktur Data Program

Bahasa pemrograman yang digunakan untuk mengembangkan bot adalah Python. Program menggunakan struktur data yang didefinisikan melalui class. Class-class yang diperlukan untuk pengembangan bot berada di dalam folder bernama `game/`, yang berada pada folder `bot starter pack`.

Ada 2 file yang terdapat pada folder game yaitu *models.py* dan *utils.py*. *models.py* terdapat beberapa class yaitu *bot*, *position*, *properties*, *gameObject*, dan *board*. *utils.py* memiliki beberapa fungsi yaitu *clamp*, *get_direction*, *position_equals*

Class yang terdapat pada *models.py* antara lain :

- *Class bot* merupakan kelas untuk melakukan instansiasi objek *bot*, yang terdiri dari 3 komponen, yaitu nama, email, dan juga id *bot*.
- *Class Position* merupakan kelas yang menampung informasi koordinat x dan y atau cenderung digunakan sebagai tipe data.
- *Class Properties* didesain untuk menyimpan berbagai informasi sebagai berikut:
 - *points* merepresentasikan jumlah poin yang diperoleh dari sebuah diamond, yang umumnya adalah 1 atau 2 poin,
 - *pair_id* digunakan untuk memasangkan *Teleport*,
 - *diamonds* merupakan jumlah *diamond* yang sedang dibawa oleh *bot*,
 - *score* adalah jumlah poin total yang telah bot kumpulkan ke dalam basenya, atau poin yang ditunjukkan pada papan skor,
 - *name* berisi informasi nama dari objek tersebut,
 - *inventory_size* menunjukkan ukuran inventory dari objek, yang untuk bot biasanya adalah 5,
 - *can_tackle* berisi informasi apakah sebuah objek bisa ditabrak atau tidak,
 - *milliseconds_left* menyatakan waktu tersisa bagi bot untuk bermain dalam arena,
 - *time_joined* menginformasikan kapan objek mulai bergabung dalam permainan,
 - *base* memberikan detail lokasi base dari objek tersebut jika ada.
- *Class GameObject* Ini adalah sebuah kelas yang menyimpan informasi seperti id, lokasi, jenis, dan properti dari suatu objek yang diciptakan berdasarkan blueprint kelas ini. Properti adalah perluasan kelas dari kelas sebelumnya..
- *Class Board* Ini adalah kelas yang menggabungkan id, dimensi lebar dan tinggi, fitur-fitur, *minimum_delay_between_moves* (yang menetapkan interval minimal antara gerakan bot), dan *game_objects* yang mencakup objek-objek yang ada dalam papan permainan. Dalam kelas ini juga termasuk fungsi seperti *is_valid_move*, yang berfungsi untuk memverifikasi keabsahan gerakan yang dibuat oleh bot.

Pada file *utils.py* yang terdapat beberapa fungsi:

- Fungsi *clamp* digunakan untuk memastikan nilai n berada dalam rentang antara nilai minimum dan maksimum. Jika n lebih rendah dari nilai minimum, maka fungsi akan mengembalikan nilai minimum itu sendiri. Sebaliknya, jika n melebihi nilai maksimum, maka akan dikembalikan nilai maksimum tersebut.
- Fungsi *get_direction* bertugas memberikan nilai delta_x dan delta_y, yang menandakan arah pergerakan bot menuju lokasi tujuan (*destination*).
- Fungsi *position_equals* berfungsi untuk memeriksa apakah dua posisi yang diberikan sebagai parameter adalah identik atau tidak.

Di dalam folder *game/logic/*, terdapat beberapa file yang mengandung logika untuk implementasi strategi greedy alternatif yang bertindak sebagai kecerdasan utama bot yang akan dimainkan. File tersebut menyertakan fungsi *next_move()*, yang mengaplikasikan strategi greedy untuk menentukan posisi tujuan (*goal_position*). Setelah itu, posisi tujuan ini akan diolah oleh fungsi *get_direction*, yang diimpor dari file *utils.py* atau fungsi lain yang telah dibuat pada suatu class bot, untuk menentukan arah gerak bot menuju posisi tujuan tersebut.

File dengan nama *HighestBlockPerDistanceBot.py* pada folder *game/logic/* merupakan file yang berisi algoritma greedy yang kami pilih. Dalam file ini terdapat class bernama *HighestBlockPerDistanceBotLogic* yang merupakan turunan dari class *BaseLogic*. Pada class ini terdapat beberapa fungsi yang dibuat. Fungsi utama yang ada tentunya adalah *next_move* yang berisi algoritma greedy utama yang digunakan. Fungsi *next_move tersebut* digunakan untuk menentukan arah gerak dari suatu bot pada suatu waktu. Fungsi *next_move* juga merupakan fungsi yang wajib dimplementasikan dan merupakan fungsi yang diturunkan dari class *BaseLogic*. Di dalam class ini juga terdapat fungsi *get_surrounding_points* untuk menghitung total poin diamond pada suatu block berukuran 3x3. Selain itu terdapat juga fungsi *get_distance* untuk menghitung jarak antar 2 objek dalam permainan dan juga ada fungsi *calculate_benefit* untuk menghitung nilai manfaat dari suatu block. Terdapat juga fungsi *handle_not_moving* yang mengatasi masalah ketika bot ngebug dan diam di tempat dan juga terdapat fungsi *is_teleporter_closer* untuk menentukan apakah dengan menggunakan teleporter jarak bisa menjadi lebih dekat. Fungsi terakhir yang dibuat adalah fungsi *get_direction_yfirst*

yang sebenarnya sama saja dengan fungsi *get_direction* pada file *utils.py* hanya saja fungsi ini lebih mengutamakan gerak secara vertikal terlebih dahulu.

Dalam class yang kami buat ini juga terdapat beberapa atribut yang digunakan antara lain yaitu directions yang berisi kemungkinan gerakan yang bisa dilakukan bot, current_direction yang berisi arah gerak bot saat ini, goal_position yang berisi tujuan bot saat ini, time_base_retreat yang bertipe boolean yang menunjukkan apakah bot perlu kembali ke base saat ini atau tidak, dan atribut yang terakhir adalah attack_other_bot yang bertipe boolean yang menunjukkan apakah bot perlu menyerang bot lain atau tidak. Untuk atribut attack_other_bot disini akan selalu bernilai false karena berdasarkan pengujian yang kami lakukan penyerangan bot lain cenderung tidak efektif.

File *main.py*, yang berlokasi di folder *game/*, mengandung program inti untuk mengintegrasikan semua komponen dalam paket awal bot ini. Logika yang telah dikembangkan di folder *game/logic/* dapat diintegrasikan dengan cara mengimpor dan menambahkannya ke dalam kontroler yang tersedia di dalam file ini.

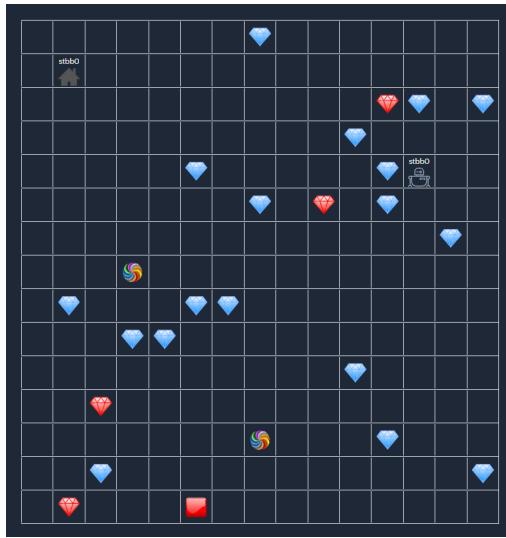
4.3 Analisis dan Pengujian

Pada bagian ini akan dilakukan pengetesan terhadap fungsionalitas program bot yang dibuat meliputi algoritma greedy utama dari bot yang dipilih dan fungsi seleksi yang telah dipilih. Dalam pengujian ini juga akan diuji beberapa kasus kapan algoritma akan memperoleh nilai optimal dan kapan algoritma tidak mendapatkan nilai optimal.

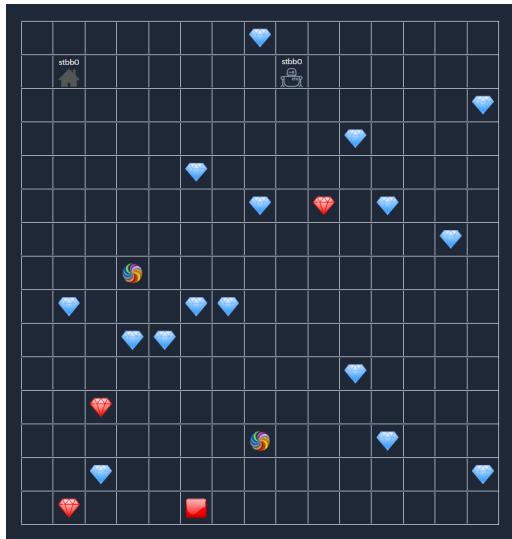
Note: posisi selalu dimulai dari angka satu dan dihitung dari pojok kiri atas

4.3.1 Kasus Algoritma Utama

Awal:



Akhir



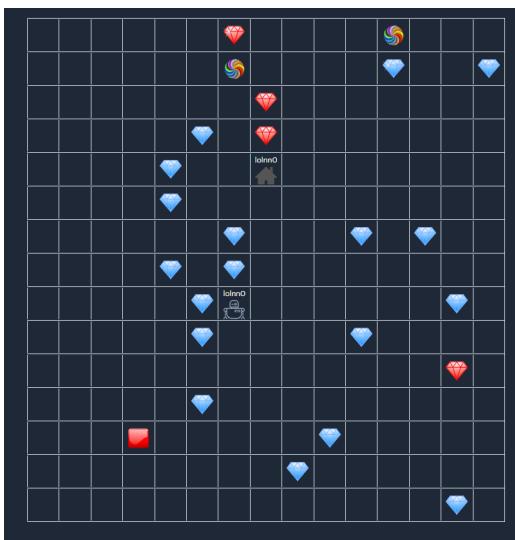
Penjelasan:

Pada kondisi awal, bot akan melakukan perhitungan terlebih dahulu untuk mencari nilai reward dari setiap block diamond per jarak ke bot yang paling besar. Setelah didapat block dengan reward per unit jarak tertinggi bot akan langsung bergerak menuju block diamond tersebut. Ketika tujuan dari bot sudah ditentukan bot baru akan mempertimbangkan apakah perlu menggunakan teleporter atau tidak. Pada kasus di atas block diamond dengan nilai reward per unit jarak tertinggi pertama adalah diamond biru yang tepat berada di sebelah kiri dari bot kemudian block diamond dengan nilai reward per unit jarak tertinggi berikutnya adalah block

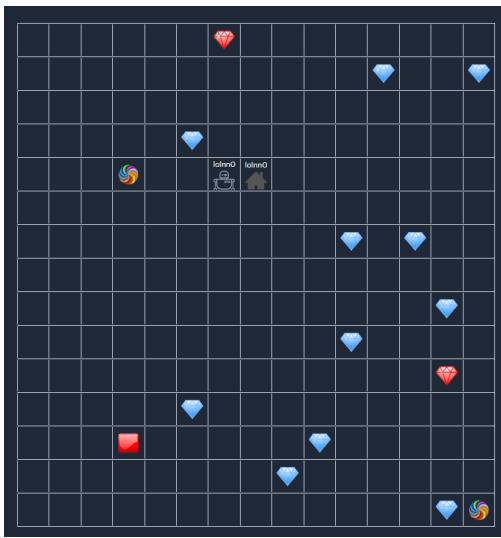
diamond yang memiliki diamond pusat berupa diamond merah di posisi (12, 3). Seperti yang terlihat pada kondisi akhir, bot telah mengambil beberapa diamond pada kedua block diamond tersebut yang menunjukkan bahwa algoritma utama telah bekerja dengan baik. Pada kasus ini jarak dari bot ke block diamond juga sangat dekat sehingga bot memutuskan untuk tidak perlu menggunakan teleporter yang jaraknya justru terlalu jauh baik dari bot ataupun dari block diamond.

4.3.2 Kasus Efektif

Awal



Akhir



Penjelasan:

Dalam kasus posisi seperti kondisi awal di atas bot akan cenderung bisa mendapatkan nilai yang sangat optimal dikarenakan terdapat banyak block diamond dengan total poin diamond yang tinggi di dekat bot dan dekat juga dengan base sehingga bot tidak akan terlalu membuang-buang waktu dalam perjalanan. Posisi base yang relatif berada di tengah-tengah board seperti ini juga memberikan keuntungan pada bot karena akan mempermudah bot tersebut untuk bisa menjangkau ke seluruh bagian dari board dengan jarak yang relatif sama.

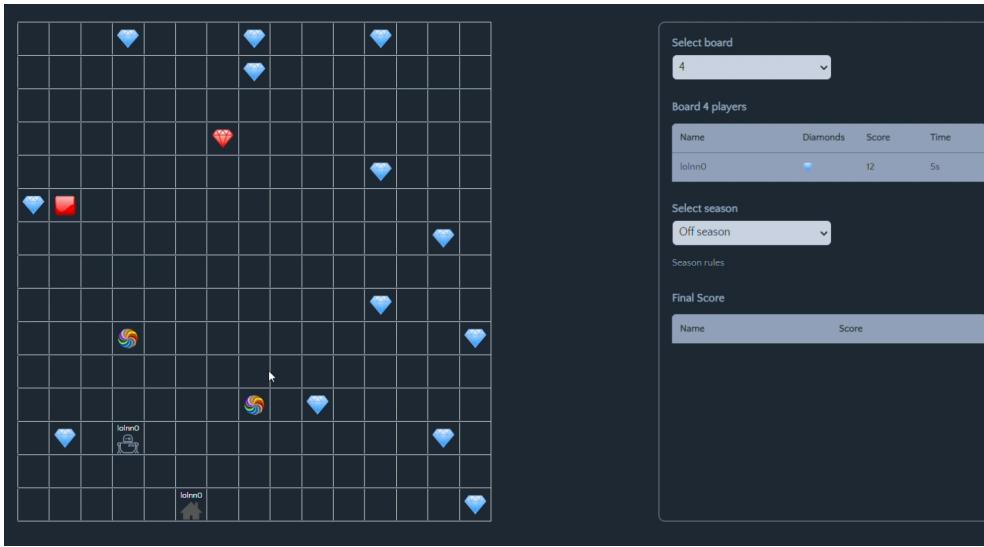
4.3.3 Kasus Tidak Efektif

Name	Diamonds	Score	Time
lolnnO	4	41s	

Name	Score
lolnnO	

Name	Diamonds	Score	Time
lolnnO	4	4	24s

Name	Score
lolnnO	

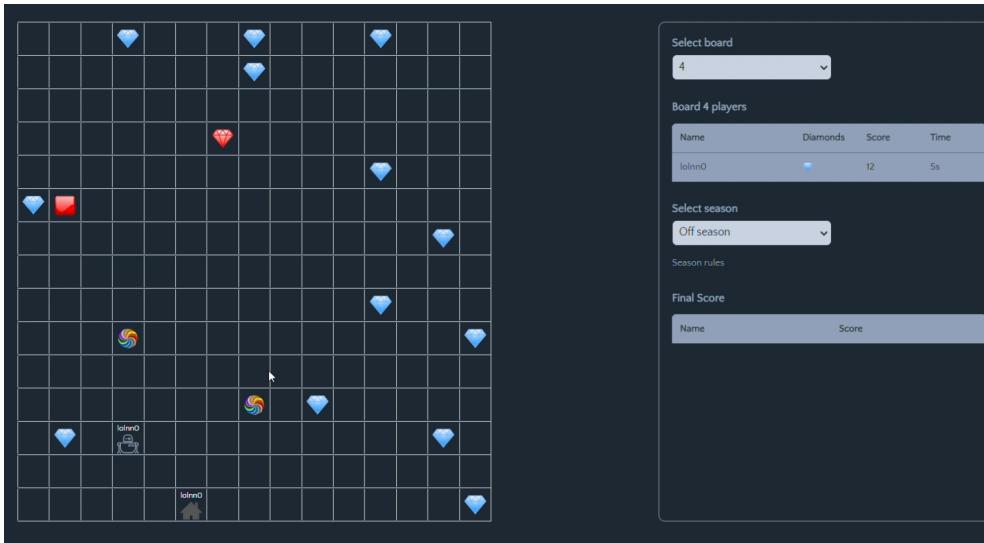


Penjelasan:

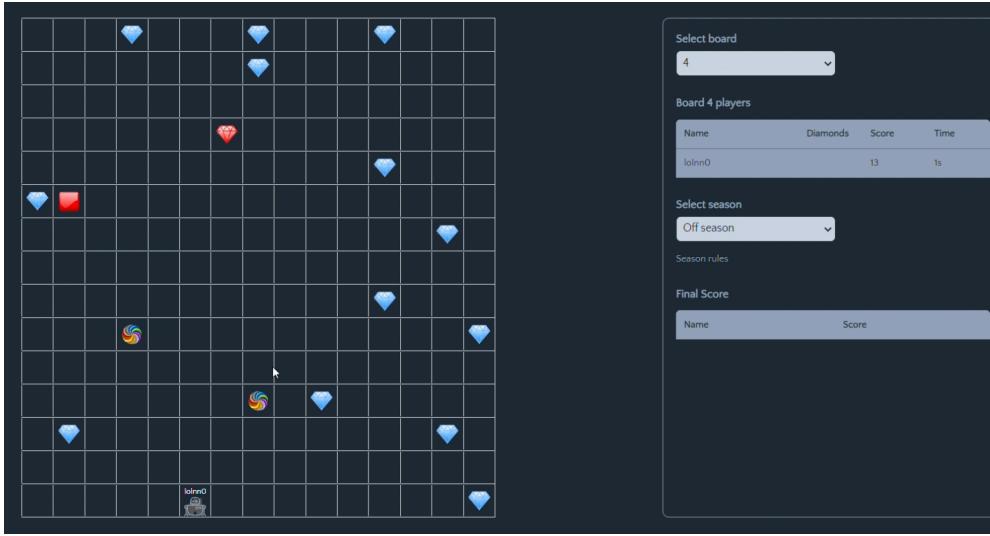
Pada kasus seperti diatas dimana diamond posisinya terlalu berjauhan dan tersebar bot ini akan cenderung kurang efektif karena perhitungan block diamond menjadi hanya didominasi oleh jarak saja. Jarak yang terlalu berjauhan juga membuat waktu bot habis diperjalanan. Pada gambar ketiga dimana posisi base berada di pojok board itu juga akan membuat bot menjadi kurang efektif dikarenakan ada daerah di board yang jaraknya terlalu jauh dari jangkauan.

4.3.4 Kasus Balik ke Base Berdasarkan Waktu

Awal:



Akhir:

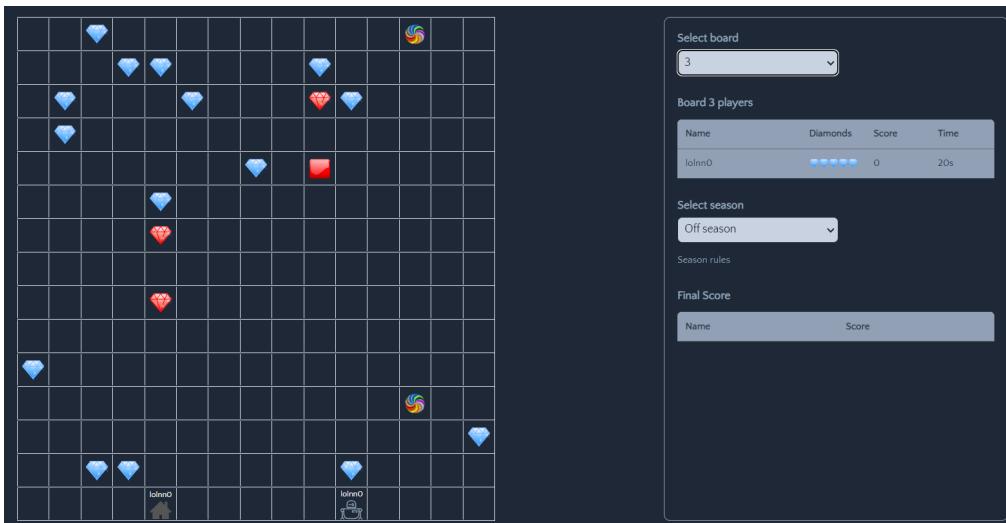


Penjelasan:

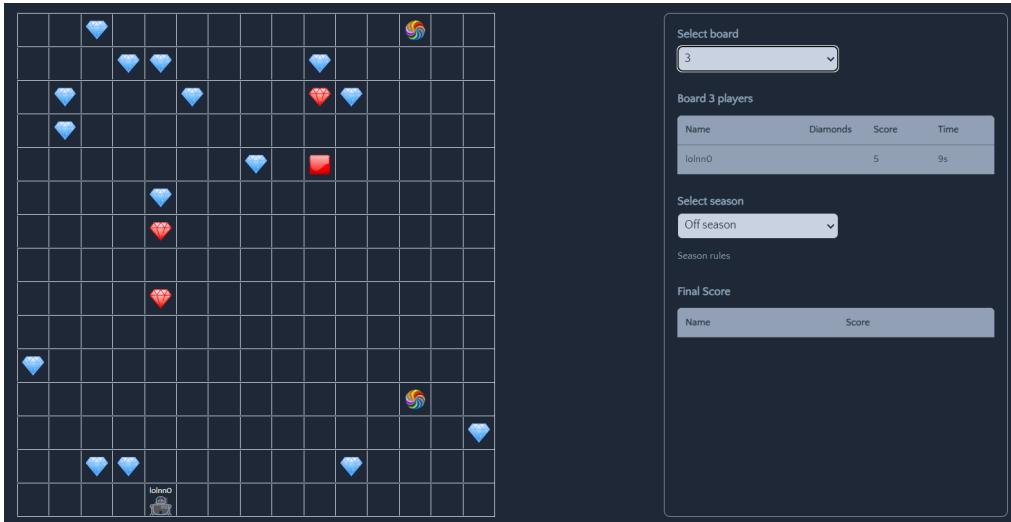
Ketika waktu yang tersisa untuk mengambil diamond sudah tidak cukup dan bot sudah membawa kurang lebih setidaknya satu diamond maka bot akan memilih untuk kembali ke base untuk mengamankan nilai dari diamond sebelum waktu habis. Pada gambar di atas pada kondisi awal waktu tersisa 5 detik dan bot sudah membawa satu diamond dikarenakan waktu sudah tidak cukup lagi maka sesuai dengan gambar pada kondisi akhir bot akan lebih memilih untuk pulang ke base dan mengamankan poin. Berdasarkan pengujian ini maka bisa dikatakan bahwa fungsi seleksi ini berjalan dengan baik.

4.3.5 Kasus Balik ke Base Kapasitas Penuh

Awal:



Akhir:

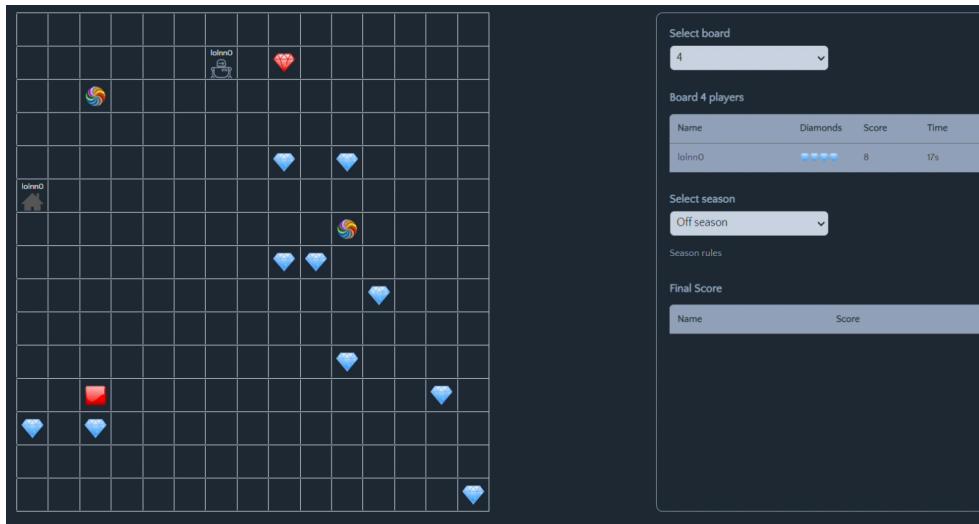


Penjelasan:

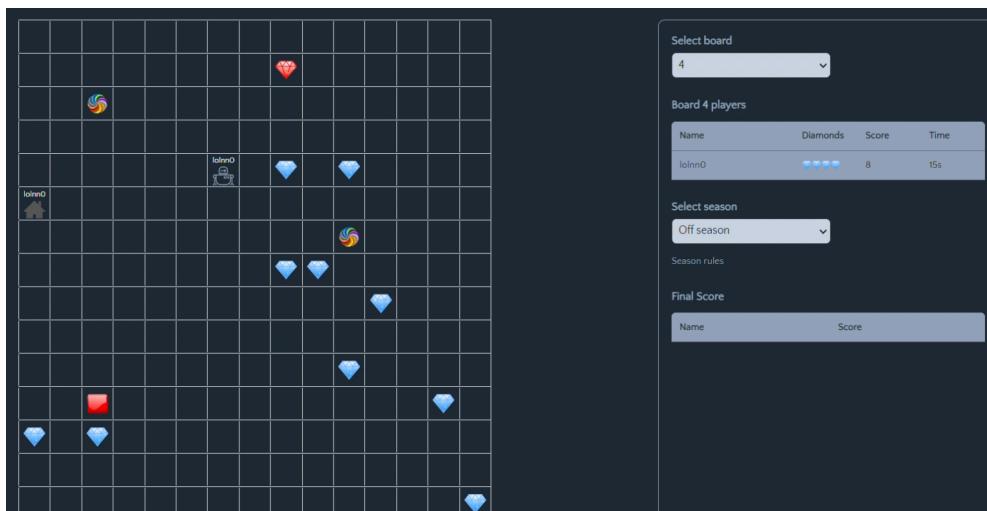
Ketika diamond yang dibawa sudah maksimal tentu saja bot akan lebih memilih untuk kembali ke base karena sudah tidak ada tempat lagi untuk membawa diamond. Pada kasus diatas bot sudah membawa 5 diamond dan dikarenakan hal tersebut bot pada akhirnya memilih untuk pulang ke base seperti yang terlihat pada kondisi akhir. Berdasarkan pengujian ini maka bisa dikatakan bahwa fungsi kelayakan ini berhasil dibuat.

4.3.6 Kasus Balik ke Base Kapasitas Sudah 4 Diamond dan Target Selanjutnya Diamond Merah

Awal:



Akhir:

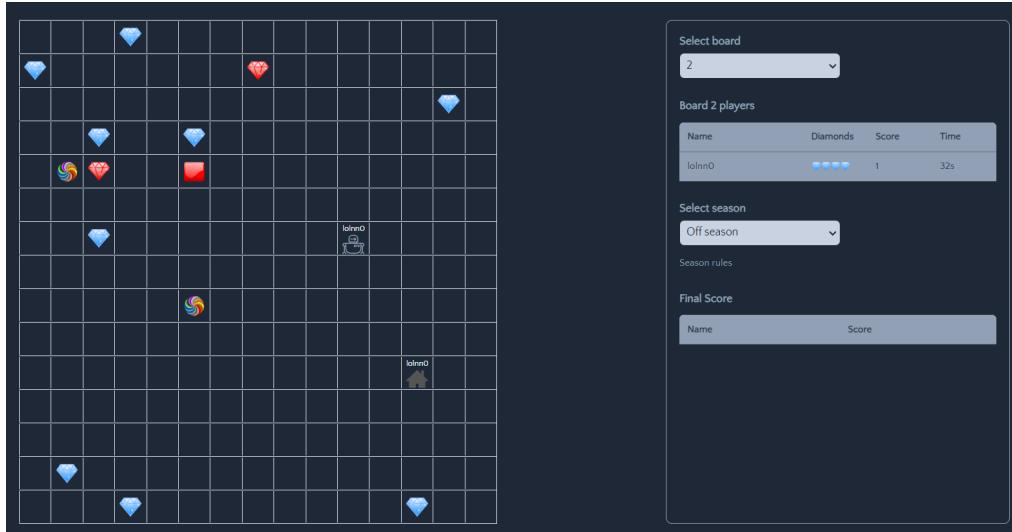


Penjelasan:

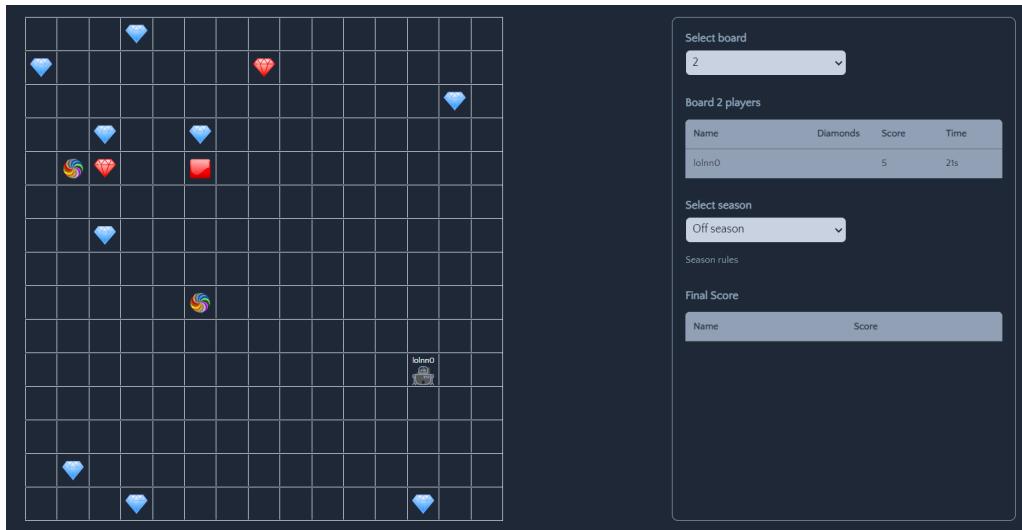
Pada gambar awal bisa dilihat bahwa total diamond yang dibawa sudah 4 dan berdasarkan kalkulasi yang dilakukan oleh bot, block diamond yang terdekat berikutnya adalah block diamond dengan diamond merah yang ada di sebelah kanannya namun karena diamond yang dibawa sudah 4 maka bot akan memilih untuk kembali ke base seperti yang terlihat pada gambar kondisi akhir. Berdasarkan pengujian ini bisa dikatakan bahwa fungsi kelayakan ini berhasil dibuat.

4.3.7 Kasus Balik ke Base Karena Jarak ke Base Lebih Dekat dari Target Selanjutnya

Awal



Akhir



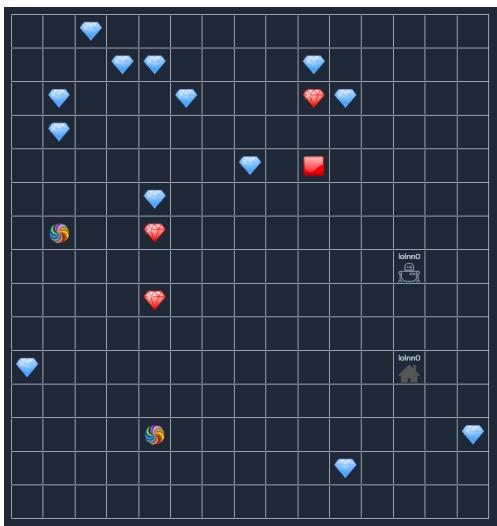
Penjelasan

Dalam kasus tersebut, bot sudah membawa lebih dari 3 diamond dan jarak dari bot ke base lebih dekat daripada jarak ke target diamond. Dalam situasi ini, bot kembali ke base untuk mengamankan poin yang telah dikumpulkan. Keuntungan dari strategi ini adalah meminimalkan risiko kehilangan poin yang sudah diperoleh, terutama jika bot berisiko diserang oleh bot lain atau menghadapi hambatan lainnya. Dengan mengutamakan pengembalian ke base, bot dapat

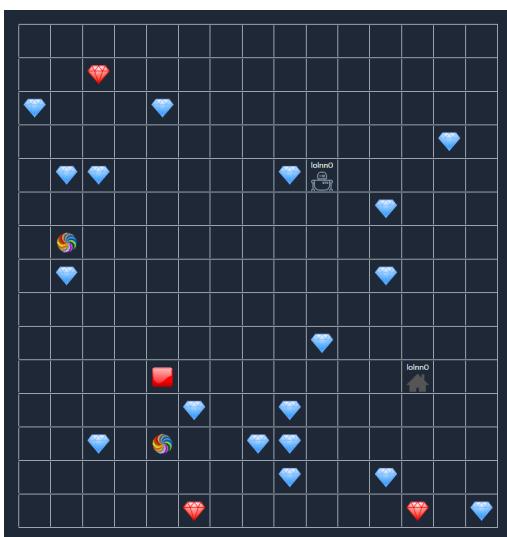
memastikan poin yang dikumpulkan tetap aman dan kemudian dapat kembali mencari diamond lain dengan lebih aman.

4.3.8 Kasus Mengambil Diamond Red Button

Awal



Akhir



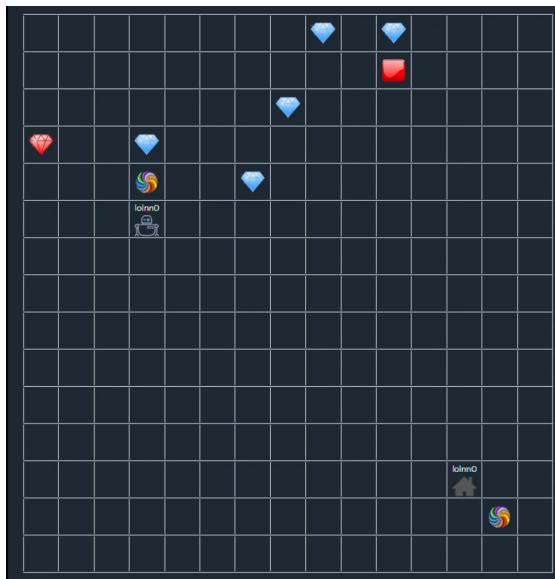
Penjelasan

Dalam kasus tersebut, jarak antara bot ke tombol diamond red button lebih dekat daripada jarak bot ke target diamond. Oleh karena itu, bot akan menuju diamond red button terlebih dahulu. Ini sangat strategis karena secara logis, jika diamond red button merupakan objek terdekat dari base, maka kemungkinan tidak ada lagi diamond yang berada dekat dengan base.

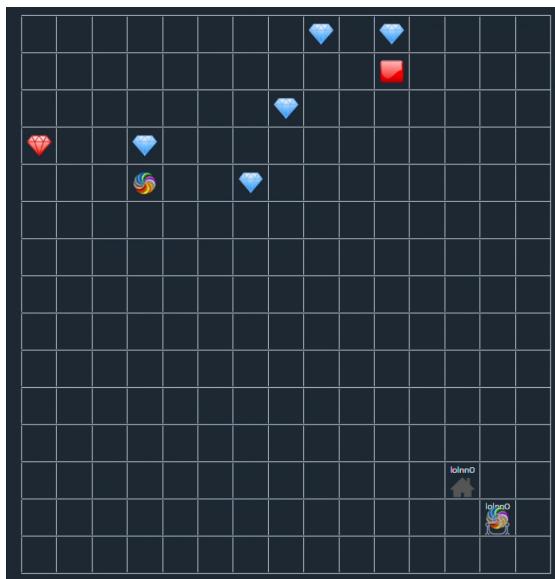
Mengaktifkan diamond red button memungkinkan untuk menghasilkan diamond baru yang dekat dengan base. Dengan demikian, hal ini memungkinkan algoritma untuk kembali menjadi efisien, yaitu ketika diamond berada di dekat base.

4.3.9 Kasus Menggunakan Teleporter

Awal



Akhir



Penjelasan

Pada kasus tersebut, tujuan bot adalah base, dalam perhitungan ternyata menggunakan teleport lebih dekat (jarak bot ke teleporter 1 ditambah jarak teleporter 2 ke base lebih kecil dari jarak bot ke base langsung). Sehingga bot akan masuk ke dalam teleporter.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pada tugas besar ini kami berhasil membuat sebuah bot untuk permainan yang menggunakan algoritma greedy. Algoritma greedy yang pada akhirnya kami gunakan untuk bot utama kami adalah algoritma greedy by *Highest Block Reward per Distance from Bot*. Algoritma ini kami pilih setelah membandingkannya dengan 11 algoritma lainnya yang telah kami temukan. Algoritma greedy pada bot ini kami pilih setelah melalui banyak tahap pengujian mulai dari pengujian fungsi kelayakan, individual test run, hingga pertandingan adu bot. Masing masing dari pengujian itu kami lakukan sebanyak 30 kali. Pada pengujian pertama kami memiliki tiga kandidat fungsi kelayakan yang kemudian fungsi kelayakan terbaik berhasil terpilih dan kami terapkan kepada semua bot yang kami buat. Pada pengujian kedua yaitu individual test run kami berhasil mendapatkan 4 bot permainan terbaik berdasarkan dari total diamond yang telah dikumpulkan dari 30 pertandingan itu. Empat bot terbaik itu adalah bot dengan algoritma greedy *shortest to bot*, *highest block reward per distance from bot*, *highest block reward per distance from bot and base*, dan *shortest to bot v2* dengan total diamond yang dikumpulkan masing-masing bot secara berurutan adalah 359, 358, 357, dan 342. Keempat bot terbaik tersebut kemudian ditandingkan pada beberapa pertandingan dan pada akhirnya didapatkan pemenang dengan poin kemenangan terbanyak yaitu algoritma greedy by highest block per distance to bot dengan poin kemenangan 63.

Berdasarkan data statistik dari pengujian yang telah kami lakukan bot dengan algoritma greedy bu highest block per distance to bot jelas adalah bot dengan algoritma greedy terbaik yang bisa kami buat untuk memenangkan permainan diamond ini. Selain menggunakan data statistik pengujian kami juga sudah mempertimbangkan pemilihan bot ini berdasarkan analisis efektifitas dan efisiensi yang telah kami lakukan.

Berdasarkan hasil uji implementasi dari algoritma greedy yang telah dipilih bisa dikatakan bahwa bot dengan *Algoritma Greedy by Highest Block Reward per Distance from Bot* ini sudah berjalan dengan baik dan optimal. Algoritma utama, fungsi kelayakan, dan fitur-fitur lainnya yang terdapat pada bot juga sudah dapat berfungsi dengan baik dan memberikan hasil yang sesuai dengan harapan yang diinginkan.

Dengan demikian dapat disimpulkan bahwa bot dengan *Algoritma Greedy by Highest Block Reward per Distance from Bot* merupakan bot terbaik berdasarkan analisis dan pengujian yang telah kami lakukan dan bot dengan algoritma tersebut juga telah berhasil kami implementasikan dengan baik dan optimal sesuai dengan aturan permainan yang ada.

5.2 Saran

1. Pengerajan laporan bisa dikerjakan lebih awal sehingga mempermudah ketika ada masalah di dekat deadline pengumpulan tugas
2. Melakukan lebih banyak pengetesan terutama dengan bot-bot dari kelompok lain sehingga lebih bisa memahami dan menemukan kelemahan dari bot yang dibuat.
3. Penggunaan teleporter dalam algoritma mungkin masih bisa lebih dimaksimalkan sehingga bisa memberikan hasil yang lebih baik lagi.
4. Perhitungan nilai reward block per jarak seharusnya bisa diberikan sebuah nilai konstanta lagi untuk lebih mendapatkan nilai normalisasi yang lebih baik lagi.

LAMPIRAN

Link Github: https://github.com/juanaw6/Tubes1_KOIN-ONDO

Link Video: <https://youtu.be/DZ6cPJ12ZbQ?feature=shared>

DAFTAR PUSTAKA

- Diamonds Game Engine. Diakses 7 Maret 2024.
<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>.
- Munir, Rinaldi. n.d. "Algoritma Greedy (Bagian 1)." Informatika. Diakses 7 Maret, 2024.
[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).
- Munir, Rinaldi. n.d. "Algoritma Greedy (Bagian 2)." Informatika. 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).
- Munir, Rinaldi. n.d. "Algoritma Greedy (Bagian 3)." Informatika. Diakses 7 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).