

LAPORAN TUGAS BESAR 1

IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma Greedy dalam Pembuatan Bot Permainan Diamonds



Oleh Kelompok ilmu padi:

- 1. Debrina Veisha Rashika W (13522025)**
- 2. Nabila Shikoofa Muida (13522069)**
- 3. Bagus Sambega Rosyada (13522071)**

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

DAFTAR ISI

DAFTAR ISI.....	1
DAFTAR GAMBAR.....	3
DAFTAR TABEL.....	4
BAB I	
DESKRIPSI MASALAH.....	5
BAB II	
LANDASAN TEORI.....	7
2.1. Algoritma Greedy.....	7
2.2. Elemen pada Greedy.....	7
2.3. Cara Kerja Permainan.....	7
2.4. Alur Menjalankan Program.....	9
2.5. Alur Pengembangan Bot.....	11
BAB III	
APLIKASI STRATEGI GREEDY.....	13
3.1. Mapping Persoalan.....	13
3.2. Alternatif Greedy.....	13
3.2.1. Greedy by Chasing Diamond.....	13
3.2.2. Greedy by Chasing Enemy.....	15
3.2.3. Greedy by Inventory.....	17
3.2.4. Greedy by Shortest Path to Current Position.....	19
3.2.5. Greedy by Closest to Base.....	21
3.3. Strategi Greedy yang Diimplementasikan.....	23
BAB IV	
IMPLEMENTASI DAN PENGUJIAN.....	25
4.1. Implementasi dalam Pseudocode.....	25
4.1.1. Initialization.....	25
4.1.2. Diamond Section.....	25
4.1.3. Bot Section.....	27
4.1.4. Red Button.....	28
4.1.5. Teleporter.....	29
4.1.6. Get Directions.....	29
4.1.7. Greedy Implementations (next_move).....	30
4.2. Struktur Data Program.....	32
4.2.1. Game.....	33
4.2.2. Logic.....	37

4.3. Analisis dan Pengujian.....	38
4.3.1. Pengujian I.....	39
4.3.2. Pengujian II.....	40
4.3.3. Pengujian III.....	41
4.3.4. Pengujian IV.....	42
4.3.5. Pengujian V.....	43
BAB V	
KESIMPULAN DAN SARAN.....	44
5.1. Kesimpulan.....	44
5.2. Saran.....	44
DAFTAR PUSTAKA.....	45
LAMPIRAN.....	46
A. Repository Github.....	46
B. Youtube Video.....	46
C. Naskah Video Youtube.....	46

DAFTAR GAMBAR

<i>Gambar 1. Ilustrasi Permainan Diamonds</i>	<i>4</i>
<i>Gambar 2. Contoh Isi File “run.bat”</i>	<i>8</i>
<i>Gambar 3. Contoh Isi File “run.bat”</i>	<i>8</i>
<i>Gambar 4. Frontend Permainan Diamonds</i>	<i>9</i>
<i>Gambar 5. Struktur File Permainan Diamonds</i>	<i>10</i>
<i>Gambar 6. Contoh Implementasi “mybot.py”</i>	<i>11</i>

DAFTAR TABEL

Tabel 1. <i>Tabel Mapping Element Greedy by Chasing Diamonds</i>	14
Tabel 2. <i>Tabel Mapping Elemen Greedy by Chasing Enemy</i>	16
Tabel 3. <i>Tabel Mapping Elemen Greedy by Inventory</i>	17
Tabel 4. <i>Tabel Mapping Elemen Greedy by Shortest Path to Current Position</i>	19
Tabel 5. <i>Mapping Elemen Greedy by Closest to Base</i>	21
Tabel 6. <i>Tabel Struktur Data “api.py”</i>	33
Tabel 7. <i>Tabel Struktur Data “board_handler.py”</i>	33
Tabel 8. <i>Tabel Struktur Data “bot_handler.py”</i>	34
Tabel 9. <i>Tabel Struktur Data “models.py”</i>	34
Tabel 10. <i>Tabel Struktur Data “util.py”</i>	36
Tabel 11. <i>Tabel Struktur Data “padibot.py”</i>	37
Tabel 12. <i>Tabel Poin Pertandingan Padibot & Kawannya</i>	38
Tabel 13. <i>Tabel Hasil Pengujian I dan Analisisnya</i>	39
Tabel 14. <i>Tabel Hasil Pengujian II dan Analisisnya</i>	40
Tabel 15. <i>Tabel Hasil Pengujian III dan Analisisnya</i>	41
Tabel 16. <i>Tabel Hasil Pengujian IV dan Analisisnya</i>	42
Tabel 17. <i>Tabel Hasil Pengujian V dan Analisisnya</i>	43

BAB I

DESKRIPSI MASALAH

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



Gambar 1. Ilustrasi Permainan Diamonds

Berikut ini adalah cara kerja dari 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 menempa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

BAB II

LANDASAN TEORI

2.1. Algoritma Greedy

Algoritma *greedy* merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Algoritma ini memecahkan persoalan secara langkah per langkah (step by step) dan menganut prinsip *take what you can get now*, yang berarti mengambil pilihan terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi kedepannya. Jadi pada setiap langkah, dipilih optimum lokal dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global [1].

2.2. Elemen pada Greedy

Setiap strategi *greedy* yang digunakan memiliki fokus utama dan pendekatan langkah yang berbeda-beda, dan dapat dideskripsikan pada elemen-elemen berikut:

- Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah.
- Himpunan solusi, S : berisi kandidat yang sudah dipilih.
- Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
- Fungsi seleksi (*selection function*): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
- Fungsi kelayakan (*feasible*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
- Fungsi obyektif : memaksimumkan atau meminimumkan.

2.3. Cara Kerja Permainan

Secara garis besar, permainan dibagi menjadi dua struktur utama, yaitu *game engine* yang merupakan program untuk menjalankan permainan *Diamonds* dan *setup environment*, dan juga *game logic* yang merupakan “otak” dari bot yang bekerja pada permainan. Untuk menjalankan bot pada permainan dapat mengunduh terlebih dahulu mengunduh *starter pack* game pada link

[bot-starter-pack](#) dan melakukan *setup* dengan melakukan instalasi *requirement* dari file “requirements.txt”.

Permainan ini merupakan permainan berbasis web, sehingga setiap aksi yang dilakukan memerlukan HTTP *request*. Pengguna perlu melakukan kompilasi program “main.py” untuk menjalankan bot dan mengirimkan fungsi *logic* bot ke server. Berikut adalah urutan *requests* pada permainan.

1. Bot akan melakukan pengecekan apakah bot telah terdaftar atau belum dengan mengirimkan permintaan POST ke titik akhir `/api/bots/recover` dengan mengirimkan email dan kata sandi bot sebagai badan permintaan. Jika bot telah terdaftar, backend akan memberikan kode respons 200 dengan badan respons yang berisi ID bot. Jika belum terdaftar, backend akan memberikan kode respon 404.
2. Jika bot belum terdaftar, bot program akan mengirimkan permintaan POST ke titik akhir `/api/bots` dengan badan permintaan yang berisi email, nama, kata sandi, dan tim. Jika berhasil, backend akan memberikan kode respons 200 dengan badan respons yang berisi ID bot tersebut.
3. Setelah mendapatkan ID bot, bot dapat bergabung ke papan permainan dengan mengirimkan permintaan POST ke titik akhir `/api/bots/{id}/join` dengan badan permintaan yang berisi ID papan yang diinginkan (`preferredBoardId`). Jika bergabung berhasil, backend akan memberikan kode respons 200 dengan badan respons yang berisi informasi tentang papan permainan.
4. Bot program akan secara berkala menghitung langkah selanjutnya berdasarkan kondisi papan yang diketahui dan mengirimkan permintaan POST ke titik akhir `/api/bots/{id}/move` dengan badan permintaan yang berisi arah langkah selanjutnya ("NORTH", "SOUTH", "EAST", atau "WEST"). Jika berhasil, backend akan memberikan kode respons 200 dengan badan respons yang berisi kondisi papan setelah langkah tersebut. Langkah ini akan terus diulang sampai waktu bot habis. Jika waktu habis, bot akan otomatis keluar dari papan.
5. Program frontend juga akan secara periodik mengirimkan permintaan GET ke titik akhir `/api/boards/{id}` untuk mendapatkan kondisi terbaru papan, sehingga tampilan papan pada frontend akan selalu diperbarui.

2.4. Alur Menjalankan Program

Berikut langkah-langkah untuk menjalankan program bot;

1. Masuk ke root directory dari project dan unduh dependencies dengan perintah “pip install -r requirements.txt”.
2. Untuk menjalankan satu bot, buka terminal dan jalankan perintah:

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

Pada contoh ini dijalankan bot dengan logic yang terdapat pada file game/logic/random.py, bisa disesuaikan dengan nama file bot yang dimiliki.

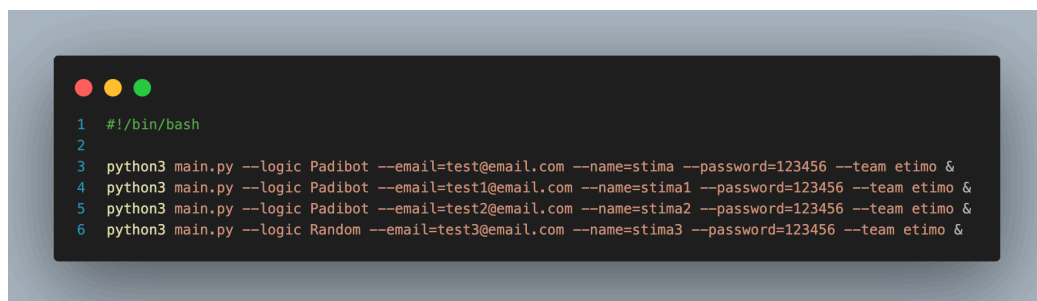
3. Pemain juga dapat menjalankan beberapa bot sekaligus dengan membuat file run.bat (khusus untuk Windows OS) sebagai berikut:



```
1 @echo off  
2 start cmd /k "python main.py --logic Padibot --email=test@email.com --name=stima --password=123456 --team etimo"  
3 start cmd /c "python main.py --logic Padibot --email=test1@email.com --name=stima1 --password=123456 --team etimo"  
4 start cmd /c "python main.py --logic Padibot --email=test2@email.com --name=stima2 --password=123456 --team etimo"  
5 start cmd /c "python main.py --logic Random --email=test3@email.com --name=stima3 --password=123456 --team etimo"  
6
```

Gambar 2. Contoh Isi File “run.bat”

Sedangkan bagi pengguna UNIX-base OS (Linux/macOS) dapat membuat file run.sh yang isinya sebagai berikut:



```
1 #!/bin/bash  
2  
3 python3 main.py --logic Padibot --email=test@email.com --name=stima --password=123456 --team etimo &  
4 python3 main.py --logic Padibot --email=test1@email.com --name=stima1 --password=123456 --team etimo &  
5 python3 main.py --logic Padibot --email=test2@email.com --name=stima2 --password=123456 --team etimo &  
6 python3 main.py --logic Random --email=test3@email.com --name=stima3 --password=123456 --team etimo &
```

Gambar 3. Contoh Isi File “run.bat”

Note: Untuk logic yang digunakan, email, nama, dan password dapat disesuaikan dengan kebutuhan.

4. Kemudian untuk menjalankan keseluruhan bot dalam file tersebut, buka terminal dan jalankan perintah:

a. Untuk Windows

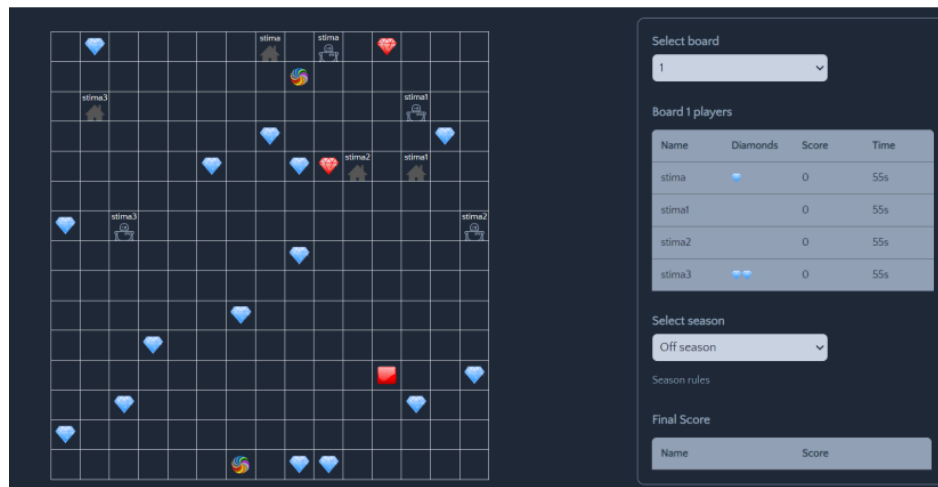
```
./run-bots.bat
```

b. Untuk UNIX-base OS (Linux/macOS)

```
chmod +x ./run-bots.sh  
./run-bots.sh
```

Note: chmod +x cukup dijalankan satu kali

5. Pastikan bot telah bergabung melalui *frontend*, seperti gambar berikut:



Gambar 4. *Frontend Permainan Diamonds*

2.5. Alur Pengembangan Bot

Program bot ini dibuat dalam bahasa pemrograman python, struktur dari file dari Bot dapat dilihat sebagai berikut:

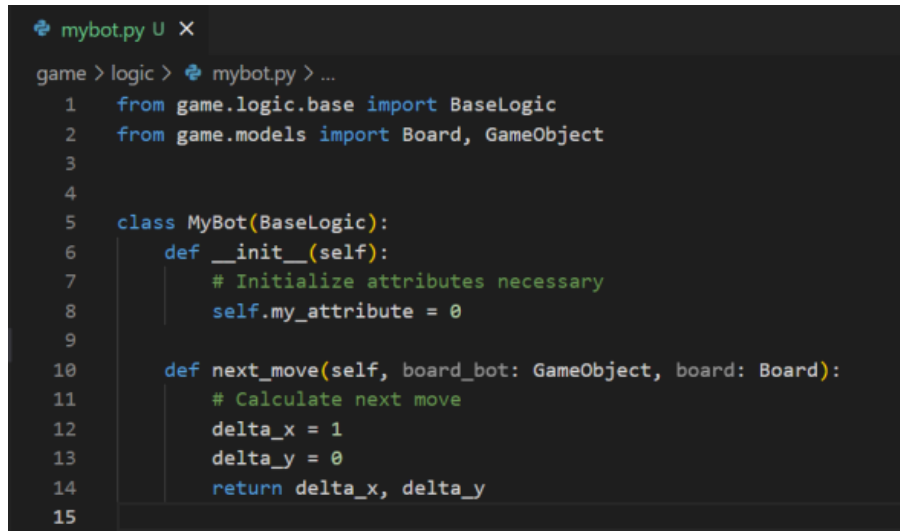


Gambar 5. Struktur File Permainan Diamonds

Pengembangan bot dapat dimulai dengan membuat file baru pada direktori `src/game/logic`. Selanjutnya buat kelas yang merupakan *inherit* dari kelas `BaseLogic`, lalu implementasikan *constructor* dan *method* “`next_move`” pada kelas tersebut. Fungsi `next_move` adalah fungsi utama yang akan mengembalikan nilai arah pada sistem permainan melalui metode *HTTP request* yang telah dijelaskan sebelumnya, berdasarkan *logic* dan algoritma yang sudah dibuat.

Fungsi `next_move` dapat memanggil fungsi-fungsi lain yang berisi *logic* dan strategi bot selama fungsi-fungsi yang dipanggil merupakan *method* kelas yang sama. Fungsi “`next_move`” mengembalikan nilai “`delta_x`” dan “`delta_y`”, di mana nilai yang diperbolehkan hanyalah (1, 0), (0, 1), (-1, 0), (0, -1) Nilai tersebut merupakan representasi dari gerak ke kanan, bawah, kiri, dan atas berturut-turut. Apabila nilai ilegal atau di-luar range board, maka move akan diabaikan oleh program.

Untuk contoh implementasi dapat dilihat sebagai berikut:



```
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
```

Gambar 6. Contoh Implementasi “mybot.py”

File yang akan dijalankan untuk melakukan koneksi dengan server adalah file “main.py” yang terletak di folder “src”. Setelah membuat *logic* bot, *import* kelas yang telah dibuat dan daftarkan pada dictionary *CONTROLLERS* di dalam file “main.py”. Jika sudah melakukan semua langkah di atas, program bisa dijalankan *command* berikut:

```
python main.py -logic [namaBot pada controller] --email=[email]
--name=[namaBot] -password=[pass] --team [namaTim]
```

dengan mengganti setiap parameter dalam kurung siku sesuai dengan nama bot yang sudah dibuat dan didaftarkan pada *controllers* di main.py, email, nama bot yang akan ditampilkan, *password*, dan nama tim.

BAB III

APLIKASI STRATEGI GREEDY

3.1. Mapping Persoalan

Secara umum, permainan *Diamonds* merupakan permainan untuk mengumpulkan *diamond* yang merupakan representasi poin sebanyak-banyaknya. Bot yang sudah mengumpulkan *diamond* pada *inventory* akan mendapatkan poin dengan kembali ke *base* dan menyimpan *diamond* tersebut, sehingga *inventory* kembali kosong dan bot mencari lagi *diamond*.

Setiap strategi *greedy* yang digunakan memiliki fokus utama dan pendekatan langkah yang berbeda-beda, dan dapat dideskripsikan sebagai berikut,

- Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah.
- Himpunan solusi, S : berisi kandidat yang sudah dipilih.
- Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
- Fungsi seleksi (*selection function*): memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi *greedy* ini bersifat heuristik.
- Fungsi kelayakan (*feasible*): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
- Fungsi obyektif : memaksimumkan atau meminimumkan.

3.2. Alternatif Greedy

3.2.1. Greedy by Chasing Diamond

Greedy by Chasing Diamond adalah strategi *greedy* yang mengutamakan pengumpulan berlian yang sebanyak-banyaknya tanpa mengindahkan strategi gerakan melalui *red button* ataupun *tackle* bot lain. Bot akan bergerak secara agresif dan hanya berfokus untuk mengambil berlian di sekitarnya dan memprioritaskan berlian merah yang memiliki poin tertinggi. Bot akan terus mencari dan mengeksploitasi peluang untuk mengumpulkan berlian dengan efisien, tanpa terlalu mempertimbangkan risiko atau strategi lain yang mungkin lebih kompleks.

➤ Mapping Elemen Greedy

Tabel 1. *Tabel Mapping Element Greedy by Chasing Diamonds*

Elemen Greedy	Analisis
Himpunan Kandidat	Seluruh pasangan aksi mengambil <i>Blue diamond</i> , mengambil <i>Red diamond</i> , atau tidak mengambil keduanya.
Himpunan Solusi	Pasangan aksi yang membuat bot memilih untuk mengambil <i>Blue diamond</i> , mengambil <i>Red diamond</i> , atau tidak mengambil keduanya yang dapat memaksimalkan <i>score</i> yang didapat.
Fungsi Solusi	Memeriksa apakah aksi yang dipilih dapat memaksimalkan jumlah poin yang dikumpulkan.
Fungsi Seleksi	Memilih aksi yang paling efisien dalam mengumpulkan <i>score</i> .
Fungsi Kelayakan	Memeriksa apakah aksi yang dipilih valid dan memenuhi batasan-batasan pergerakan bot serta dapat memaksimalkan poin yang didapat.
Fungsi Obyektif	Mengoptimalkan aksi yang akan mengambil berlian terbaik yang membuat bot memperoleh <i>score</i> paling optimal.

➤ Analisis Efisiensi Solusi

- Pada himpunan kandidat, terdapat 3 aksi yang perlu dibandingkan yaitu mengambil berlian biru, berlian merah, atau tidak mengambil keduanya. Apabila aksi yang diambil memungkinkan menambah skor bot dengan rute terdekat, maka pilih aksi tersebut. Pemilihan aksi diprioritaskan pada jarak bot terhadap berlian dan skor yang akan didapatkan. Sebelum

melakukan pengecekan akan dicari jarak berlian merah dan biru terdekat dengan bot. Kompleksitas dari tahap ini adalah $O(n)$ untuk penghitungan jarak, dan $O(n)$ pula untuk membandingkan jarak.

- Pertama akan dilakukan pengecekan apakah jumlah *inventory* bot masih mencukupi untuk mengambil berlian. Pengecekan ini dilakukan agar pengambilan berlian tidak dilakukan dengan sia-sia. Kompleksitas dari tahap ini adalah $O(1)$.
- Kedua akan dilakukan pengecekan perbandingan antara jarak berlian biru dan berlian merah terdekat. Jika berlian merah lebih dekat akan dipilih berlian merah karena skor yang didapat akan lebih besar. Kompleksitas dari tahap ini adalah $O(1)$.
- Kompleksitas waktu algoritma ini secara umum adalah $O(n)$.

➤ Analisis Efektivitas Solusi

- Strategi ini efektif jika terdapat banyak berlian yang terkumpul di sekitar bot dan bot mampu mengumpulkannya dengan cepat.
- Keefektifan strategi ini juga tergantung pada banyak rintangan yang ada dalam peta dan cara bot mengambil keputusan yang optimal untuk mencapai berlian.
- Namun, strategi ini kurang efektif jika lokasi antar *diamond* sangat berjauhan, ataupun terdapat banyak objek lain seperti *teleporter*, *red button* atau bot lawan yang menghalangi akses ke berlian.

3.2.2. Greedy by Chasing Enemy

Greedy by Chasing Enemy adalah strategi *greedy* yang menekankan serangan terus menerus terhadap lawan untuk mengurangi jumlah berlian yang dimilikinya. Bot akan berusaha melakukan *tackle* ke lawan untuk menambah pundi-pundi berlian dari bot musuh. Bot musuh yang dipilih untuk dikejar dapat didasarkan pada jumlah *diamond* yang dimiliki, ataupun jarak ke bot tersebut.

➤ Mapping Elemen Greedy

Tabel 2. *Tabel Mapping Elemen Greedy by Chasing Enemy*

Elemen Greedy	Analisis
Himpunan Kandidat	Seluruh pasangan aksi untuk <i>mentackle</i> bot lawan yang mungkin.
Himpunan Solusi	Pasangan aksi yang dapat <i>mentackle</i> bot lawan yang menyebabkan bertambahnya jumlah berlian yang dimiliki.
Fungsi Solusi	Memeriksa apakah aksi yang dipilih berhasil mengambil berlian yang dimiliki oleh bot lawan.
Fungsi Seleksi	Memilih aksi yang paling mungkin untuk berhasil dalam mencuri berlian milik bot lawan.
Fungsi Kelayakan	Memeriksa apakah aksi yang dipilih memenuhi restriksi-restriksi dan kondisi untuk dilakukan saat itu.
Fungsi Obyektif	Mencari aksi yang membuat bot dapat <i>mentackle</i> bot lain dan mencuri diamond yang dimiliki secara optimal.

➤ Analisis Efisiensi Solusi

- Pada algoritma ini akan dilakukan pencarian bot lawan yang memiliki jumlah berlian yang banyak atau lebih banyak daripada dimiliki oleh bot saat ini. Kompleksitas dari tahap ini adalah $O(n)$.
- Selanjutnya akan dilakukan pengecekan terhadap jarak setiap bot lawan yang memiliki berlian dengan bot yang dimiliki. Jika jarak bot lawan dengan bot kurang dari 3 maka bot akan memilih mengejar bot lawan tersebut. Kompleksitas dari tahap ini adalah $O(n)$.

- Kompleksitas waktu algoritma ini secara umum adalah $O(n)$.
- Analisis Efektivitas Solusi
 - Efektivitas strategi ini sangat relatif terhadap strategi bot musuh dan juga kondisi pergerakan bot musuh. Hal ini disebabkan jika bot musuh memiliki strategi untuk bergerak menjauh dari bot lain, atau memang sedang bergerak menjauh dari bot, maka proses pengejaran dapat memakan waktu sangat lama.
 - Strategi ini akan efektif jika jarak antar-bot yang akan dikejar sangat rapat dan bot tidak bergerak menjauh dari pengejaran, dengan asumsi bot yang dikejar pastilah memiliki *diamond* yang bernilai untuk dikejar.

3.2.3. Greedy by Inventory

Greedy by Inventory adalah strategi *greedy* yang digunakan untuk memaksimalkan *inventory* yang dimiliki oleh bot sebelum kembali ke *base*. Strategi ini digunakan agar tidak memakan waktu dalam melakukan pencarian berlian dan bisa sekaligus mendapat skor banyak saat menaruh berlian ke *base*.

➤ Mapping Elemen Greedy

Tabel 3. *Tabel Mapping Elemen Greedy by Inventory*

Elemen Greedy	Analisis
Himpunan Kandidat	Seluruh pasangan aksi yang digunakan untuk bot memenuhi <i>inventory</i> yang dimiliki.
Himpunan Solusi	Kemungkinan aksi yang digunakan bot untuk mencari berlian sampai <i>inventory</i> yang dimilikinya mencapai maksimum.

Fungsi Solusi	Melakukan pengecekan apakah aksi yang digunakan bisa memaksimalkan <i>inventory</i> yang dimiliki.
Fungsi Seleksi	Memilih aksi mencari berlian yang dapat membuat <i>inventory</i> menjadi penuh sebelum kembali ke <i>base</i> .
Fungsi Kelayakan	Memeriksa apakah aksi yang dipilih merupakan aksi yang valid dan berlian yang diambil telah mencukupi <i>inventory</i> yang dimiliki.
Fungsi Obyektif	Mencari aksi yang membuat <i>inventory</i> yang dimiliki dapat digunakan secara maksimum sebelum kembali ke <i>base</i> .

➤ Analisis Efisiensi Solusi

- Strategi ini dilakukan dengan memeriksa jumlah berlian yang ada di *inventory* saat ini. Jika belum penuh akan dilakukan pencarian berlian. Kompleksitas pada saat melakukan pemeriksaan *inventory* adalah $O(1)$.
- Setelah memeriksa keadaan *inventory* yang belum penuh, bot akan mencari berlian terdekat tanpa memikirkan lokasinya entah itu jauh atau dekat dari *base*. Kompleksitas pada tahap ini adalah $O(n)$.
- Setelah mencari berlian akan dilakukan pemeriksaan kembali jika berlian yang ada di *inventory* sudah penuh, bot akan diarahkan kembali ke *base* untuk mengosongkan *inventory* yang dimiliki. Kompleksitas pada tahap ini adalah $O(1)$.
- Secara umum, kompleksitas algoritma yang diperlukan untuk setiap langkahnya adalah $O(n)$.

➤ Analisis Efektivitas Solusi

- Keefektifan strategi ini bergantung dengan lokasi berlian jika banyak berlian sekitar *base* skor yang didapat akan semakin besar. Sedangkan jika jarak antar berlian berjauhan waktu yang digunakan akan lebih banyak.
- Strategi ini juga kurang efektif jika banyak lawan di sekitar bot yang ingin melakukan penyerangan saat bot sedang membawa banyak berlian dan lokasi ke *base* pada saat itu cukup jauh.

3.2.4. Greedy by Shortest Path to Current Position

Greedy by Shortest Path to Current Position adalah strategi *greedy* yang mengutamakan kecepatan dalam mengambil dan menyimpan berlian, baik ke lokasi berlian ataupun bot musuh terhadap lokasi bot kami. Bot akan bergerak untuk mencari lokasi *diamond* terdekat ataupun bot musuh dengan lokasi yang dekat untuk dilakukan *tackle*. Strategi ini mempertimbangkan jarak antar-*item* dan *game objects* lain pada peta permainan relatif terhadap lokasi bot saat ini.

➤ Mapping Elemen Greedy

Tabel 4. *Tabel Mapping Elemen Greedy by Shortest Path to Current Position*

Elemen Greedy	Analisis
Himpunan Kandidat	Seluruh pasangan aksi lokasi berlian yang tersedia di peta permainan dan lokasi bot musuh terhadap lokasi bot kita.
Himpunan Solusi	Pasangan aksi yang membuat bot memilih rute optimal untuk mencapai lokasi berlian terdekat atau bot musuh dengan lokasi yang dekat untuk dilakukan tackle.

Fungsi Solusi	Memeriksa apakah rute yang dipilih dapat mencapai lokasi berlian terdekat atau bot musuh dengan efisien.
Fungsi Seleksi	Memilih rute yang paling efisien dalam mencapai lokasi berlian terdekat atau bot musuh untuk dilakukan tackle.
Fungsi Kelayakan	Memeriksa apakah rute yang dipilih memenuhi batasan-batasan pergerakan bot dan dapat mencapai lokasi berlian terdekat atau bot musuh dengan efisien.
Fungsi Obyektif	Mengoptimalkan rute yang akan mencapai lokasi berlian terdekat atau bot musuh dengan efisien, baik dalam pengambilan berlian maupun dalam melakukan tackle terhadap bot musuh.

➤ Analisis Efisiensi Solusi

- Strategi ini hanya mempertimbangkan jarak bot ke *diamonds* dan bot lain, sehingga bot tidak perlu mengambil jumlah langkah yang terlalu banyak untuk mencari *diamonds* atau bot lain. Jika diasumsikan banyak *object* yang ada di peta permainan adalah n , maka kompleksitas algoritma yang dibutuhkan untuk mencari jarak item ke lokasi saat ini adalah $O(n)$, ditambah dengan pencarian nilai minimum dengan kompleksitas $O(n)$.
- Strategi ini akan efisien jika lokasi *diamonds* ataupun bot musuh yang akan dikejar memiliki jarak dekat dengan lokasi bot saat ini. Setelah ditemukan titik terdekat dengan jarak m langkah, maka kompleksitas pergerakan adalah $O(m)$, kemudian untuk kembali ke *base* dengan jarak p langkah, kompleksitasnya adalah $O(p)$.

- Secara umum, kompleksitas algoritma yang diperlukan untuk setiap langkahnya adalah $O(n)$.
- Analisis Efektivitas Solusi
 - Keefektifan strategi ini bergantung pada struktur dan distribusi objek dalam peta permainan karena sangat mempertimbangkan jarak relatif antara posisi bot saat ini dengan lokasi berlian ataupun bot musuh terhadap lokasi bot kami.
 - Strategi ini sangat efektif apabila terdapat berlian atau bot musuh yang dekat dengan bot kami saat ini sehingga bot dapat dengan cepat mencapai target atau menyerang musuh tanpa membuang waktu berlebihan. Strategi ini pun cukup efektif apabila berlian atau bot musuh tersebar luas dan berada dalam jarak yang relatif jauh dari posisi bot karena bot akan memilih untuk mengambil atau mengejar target yang lebih dekat terlebih dahulu.

3.2.5. Greedy by Closest to Base

Greedy by Closest to Base adalah strategi *greedy* yang berfokus pada pencarian berlian dan sumber berlian lain (misalnya dari bot lawan) terdekat dengan lokasi *base*. Strategi ini memprioritaskan perlindungan berlian yang telah dikumpulkan. Bot dengan strategi ini akan berusaha keras untuk mengambil berlian dengan jarak aman dari *base* dan mempertahankan berliannya dengan kembali ke *base*-nya sesering mungkin.

➤ Mapping Elemen Greedy

Tabel 5. *Mapping Elemen Greedy by Closest to Base*

Elemen Greedy	Analisis
Himpunan Kandidat	Seluruh kombinasi aksi yang bot pemain kembali ke base dalam waktu tertentu.

Himpunan Solusi	Kemungkinan aksi yang membuat bot segera kembali ke base untuk menaruh berlian dan mengosongkan inventory.
Fungsi Solusi	Melakukan pengecekan apakah aksi yang diambil dapat membuat bot kembali ke base dan dapat menambah skor dengan menaruh berlian yang dimiliki.
Fungsi Seleksi	Memilih aksi yang berdasarkan keadaan inventory dan jarak bot ke base pada saat itu yang dapat memaksimalkan skor yang didapat oleh bot.
Fungsi Kelayakan	Memeriksa apakah aksi yang dipilih merupakan aksi yang valid dan membuat bot dapat memaksimalkan skor yang didapat dengan kembali ke base.
Fungsi Obyektif	Mencari aksi yang membuat bot dapat kembali ke <i>base</i> dan mengosongkan <i>inventory</i> yang dimiliki dan mengoptimalkan skor yang akan didapat.

➤ Analisis Efisiensi Solusi

- Pada *greedy by Closest to Base*, akan diprioritaskan pencarian berlian dan sumber berlian lain (misalnya dari bot lawan) terdekat dengan lokasi base. Setelah mendapat lokasi berlian atau sumber lain yang dekat dengan base, bot akan diarahkan untuk bergerak ke objek tersebut. Kompleksitas dari tahap pencarian ini adalah $O(n)$.
- Jika saat pencarian berlian, jarak bot dan *base* adalah kurang dari sama dengan 2 kotak maka bot akan kembali ke base terlebih dahulu untuk

menaruh berlian dan mengosongkan *inventory*. Kompleksitas dari tahap ini adalah $O(1)$.

- Jika tidak ada berlian yang ada di sekitar base, akan dilakukan pemeriksaan berlian terdekat di luar base dengan kompleksitas waktu $O(n)$ dan jika jumlah *inventory* yang dimiliki sudah lebih dari tiga berlian, maka bot akan diarahkan untuk segera kembali ke base dengan kompleksitas waktu $O(1)$.
- Selain itu, juga dilakukan pemeriksaan teleporter jika jarak teleporter dekat dengan *base* maka untuk kembali ke *base* akan bot akan melewati teleporter agar waktu yang diperlukan untuk kembali ke *base* lebih cepat. Kompleksitas dari tahap ini adalah $O(1)$.
- Secara umum, kompleksitas algoritma yang diperlukan untuk setiap langkahnya adalah $O(n)$.

➤ Analisis Efektivitas Solusi

- Strategi ini akan efektif jika lokasi *diamonds* dan bot lawan banyak terletak di sekitar *base* sendiri. Sebaliknya strategi ini kurang efisien jika lokasi *diamonds* tersebar jauh dari lokasi *base*.
- Strategi ini adalah salah satu strategi “bermain aman” dengan tetap berada di jarak dekat dengan titik aman bot, yaitu *base*-nya sendiri.

3.3. Strategi Greedy yang Diimplementasikan

Berdasarkan alternatif *greedy* pada 3.2, dapat dilihat bahwa terdapat beberapa alternatif *greedy* dengan efisiensi dan efektivitasnya masing-masing. Kelompok kami memiliki prioritas untuk selalu bermain aman dengan mengimplementasikan *greedy by Closest to Base* sebagai prioritas utama bot kami. Kami menggabungkan beberapa alternatif *greedy* agar program bot dapat menangani seluruh kemungkinan kasus dari permainan Diamond secara efektif dan tepat sasaran, antara lain *greedy by Shortest Path to Current Position* dan *greedy by Chasing Diamonds*. Implementasi dari strategi kami terdapat pada file “src/game/logic/padibot.py”. Poin penting dari implementasi bot sebagai berikut:

1. Utamakan *greedy by Closest to Base* karena akan memberikan keuntungan lebih besar daripada yang lain. Bot akan mencari *diamond* ataupun bot lawan yang berada di sekitar *base*. Hal ini disebabkan karena semakin dekat dengan *base*, jarak yang dibutuhkan untuk mengembalikan berlian semakin kecil sehingga skor yang didapat akan semakin besar. Selain itu, jika berjalan terlalu jauh dari *base* besar kemungkinan berlian yang dimiliki dicuri oleh bot lawan dan membutuhkan langkah bolak-balik yang lebih banyak lagi.
2. Jika tidak ada solusi yang memenuhi fungsi kelayakan pada *greedy by Closest to Base*, maka akan dipilih aksi *greedy by Shortest Path to Current Position* dengan mencari berlian yang dekat dengan posisi bot pada saat itu walaupun posisi berlian jauh dari *base* dan tidak ada objek yang ada di sekitar *base*.
3. Jika tidak ada berlian yang dekat dengan bot dan posisi berlian jauh dari base maupun posisi bot pada saat itu maka akan digunakan pendekatan *greedy by Chasing Diamonds* untuk mengambil berlian yang terletak sangat jauh dan pada saat itu posisi berlian memang sangat jauh dari manapun.
4. Jika tidak ada berlian yang dekat maupun bot yang memenuhi untuk dikejar, maka bot akan menuju *red button* jika jaraknya lebih dekat dibandingkan *diamond* terdekat untuk *reset diamonds* yang ada dalam permainan. Hal ini dikarenakan saat *red button* dikunjungi bot, posisi *diamond* akan kembali teracak dan jumlahnya juga akan di-*generate* kembali sampai jumlah maksimum, meningkatkan kesempatan bot untuk mencari *diamond* lain.
5. Saat bot sudah memiliki cukup *diamonds*, bot akan kembali ke *base*. Saat bot menuju *base*, jika masih ada slot kosong pada *inventory* dan melewati suatu area yang masih memiliki *diamonds*, bot akan menuju *diamond* tersebut. Lalu jika terdapat *teleporter* yang dapat mempersingkat jarak tempuh bot ke *base*, bot akan memakai *teleporter* tersebut, jika tidak, bot akan langsung menuju ke *base*.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi dalam *Pseudocode*

4.1.1. Initialization

```
{ Fungsi untuk inisialisasi objek }  
function initialize():  
    directions <- [(1, 0), (0, 1), (-1, 0), (0, -1)]  
    goal_position <- None { Posisi yang menjadi tujuan }  
    current_direction <- 0  
    chaseSteps <- 0
```

4.1.2. Diamond Section

```
{ fungsi mengambil lokasi kumpulan diamond yang ada di sekitar base }  
function diamondsaroundbase(board_bot, board):  
    props <- properties of board_bot  
    diamonds <- diamond in board  
    diamond_loc <- empty []  
    for each diamond in diamonds:  
        if diamond exist in area 3 x 3 around base:  
            { Hanya akan menambahkan diamond di area berukuran 3 x 3 di sekitar base }  
            add diamond position to diamond_loc  
    -> diamond_loc  
  
{ memeriksa apakah lokasi bot kita lagi di sekitar base (area 3 x 3) }  
function botaroundbase(board_bot):  
    props <- properties of board_bot  
    current_position <- position of board_bot  
    -> true if bot in area 3 x 3 from base  
  
{ mencari diamond yang paling dekat base dengan posisi bot }  
function closestdiamondbase(board_bot, diamonds):  
    current_position <- position of board_bot  
    closest_diamond <- min(distance(diamond from diamonds,  
current_position))  
    { Return diamond dengan jarak paling minimum ke bot }
```

```
-> closest_diamond

{ cek apakah ada diamond sekitar base }
function cekdiamondbase(board_bot, board):
    diamonds <- board.diamonds
    // lokasi diamonds sekitar base
    for each diamond in diamonds:
    { Return true jika diamond ada di area 4 x 4 sekitar base }
        if diamonds exist in range -2 < position around base < 2:
            -> True
    -> False

{ cari diamond terdekat dengan bot (bebas di mana aja) }
function closestdiamond(board_bot, board):
    current_position <- position of board_bot
    closest_diamond <- minimum(positive(position of diamond -
board_bot position))
    -> position of closest_diamond

{ cari red diamond terdekat dengan bot (bebas di mana aja) }
function closestreddiamond(board_bot, board):
    current_position <- position of board_bot
    red_diamonds <- diamond in board if diamond points == 2
    if red_diamonds empty:
        -> None { Jika tidak ada red diamond, kembalikan None }
    closest_red_diamond <- min(distance(red_diamonds,
current_position))
    { Temukan red diamond terdekat dari posisi bot }
    -> position closest_red_diamond

{ Menghitung jarak diamond terdekat }
function closestdiamondddist(board_bot, board):
    closest <- closestdiamond(board_bot, board)
    current_position <- position of board_bot
    -> positive(distance(current_position, closestdiamond position))

{ mencari jarak bot dengan diamond merah terdekat }
function closestreddiamondddist(board_bot, board):
    current_position <- position of board_bot
    red_diamonds <- list of diamond in boards if point of diamond==2
```

```
if list of red_diamonds is empty:
    return None { Jika tidak ada red diamond, kembalikan None }
closest_red_diamond <- min(distance(red_diamonds,
current_position)) { Mencari red diamond terdekat }
distance <- positive(closest_red_diamond - current_position )
-> distance

{ mencari jarak bot dengan base }
function basedistance(board_bot):
    // mencari jarak bot dengan base
    current_position <- position of board_bot
    base <- board_bot
    -> positive(distance(current_position, base position))
```

4.1.3. Bot Section

```
{ Menghitung jarak (x, y) dari enemy bot ke kita }
function calculateDistanceToBots(board_bot, enemy_bot):
    -> (enemy_bot x position - position of board_bot.x, enemy_bot y
position - y position of board_bot)

{ Mencari semua bot yang memiliki diamonds >= 3 saja dan diamonds nya
lebih banyak dari kita }
function findAllBots(board_bot, board):
    listBots <- []
    for each bot in board.bots:
        { Mencari bot lain yang bukan bot kita }
        if (bot.id ≠ board_bot.id):
            if (bot's diamonds > board_bot's diamonds and bot's
diamonds ≥ 3):
                add bot to listBots
    -> listBots

{ Menjadikan bot sebagai goal_position, jika jarak ke bot musuh adalah
3, dan jarak bot kita ke base tidak lebih dari 6 }
function chaseBots(board_bot, board):
    if basedistance(board_bot) ≤ 4 and chaseSteps ≤ 5:
        { Hanya kejar hingga 5 kali, agar tidak endless chase }
        listBots <- findAllBots(board_bot, board)
```

```
for each bot in listBots:
    X, y <- calculateDistanceToBots(board_bot, bot)
    if (x == 0 and y == 0):
        { Bot sudah berhasil di-tackle, kembali ke base.
        Konsiderasi karena jumlah diamond banyak setelah tackle }
        goal_position <- base board_bot
        -> False
    else if (x ≤ 3 and y ≤ 3):
        goal_position <- bot.position
        -> True
    else:
        goal_position <- None
        -> False
else: { Jika tidak ada bot yang memenuhi kriteria, cari goal lain}
goal_position <- None
chaseSteps <- 0
-> False
```

4.1.4. Red Button

```
{ Mencari red button }
function findRedButton(board):
    for each item in game_objects of board:
        if item type is "DiamondButtonGameObject": { tipe data sesuai }
            -> item

{ Compare jarak closest diamond dengan red button. Lebih baik generate
ulang daripada ke diamond jauh. Hanya jika diamond yang ada di board
sedikit (pre-checked in next_move) }
function compareClosestDiamondToRedButton(board_bot, board):
    if calculateDistanceRedButton(board_bot, board) <
closestdiamonddist(board_bot, board):
        -> True
    -> False

{ Menghitung jarak ke red button }
function calculateDistanceRedButton(board_bot, board):
    redButton <- findRedButton(board)
{ ambil red button }
-> positive(distance between red button and board_bot)
```

4.1.5. Teleporter

```
{ Mencari semua teleporter yang ada }  
function findAllTeleporter(board_bot, board):  
    teleporters <- list of item in board if item type is  
    "TeleportGameObject"  
    { mencari semua teleporter }  
    -> sort(teleporter in teleporters, by: distance to  
    current_location)  
    { sortir teleporter, indeks 0 adalah teleporter terdekat }  
  
    { Mencari jarak dari teleporter ke base }  
procedure goToBaseWithTeleporter(board_bot, board):  
    closestTeleporter, farthestTeleporter <-  
    findAllTeleporter(board_bot, board)  
    distToBase <- distance between farthestTeleporter with board_bot  
    distToBot <- distance between closestTeleporter with board_bot  
    if (distToBase + distToBot < basedistance(board_bot)):  
        { Jika jarak perjalanan bot ke teleporter + teleporter ke base  
        lebih kecil dari perjalanan langsung bot ke base }  
        goal_position <- position closestTeleporter
```

4.1.6. Get Directions

```
{ Set direction and move, 2nd main function }  
function get_directions(current_x, current_y, dest_x, dest_y):  
    delta_x <- positive(dest_x - current_x)  
    delta_y <- positive(dest_y - current_y)  
    x <- 0  
    y <- 0  
  
    if (dest_x - current_x) < 0:  
        x <- -1 { jalan ke kiri }  
    else:  
        x <- 1 { jalan ke kanan }  
  
    if (dest_y - current_y) < 0:  
        y <- -1 { jalan ke bawah }  
    else:
```

```
y <- 1 { jalan ke atas }

if delta_x >= delta_y:
    dx <- x
    dy <- 0
else:
    dy <- y
    dx <- 0
-> (dx, dy)
```

4.1.7. Greedy Implementations (*next_move*)

```
{ Langkah selanjutnya yang bot lakukan }
function next_move(board_bot, board):
    props <- properties of board_bot
    current_position <- position of board_bot
    { Jika waktu tersisa hanya sedikit, lebih baik diam di base }
    if time left == distance board_bot to base:
        goal_position <- base of board_bot
        { Saat sedang beards di dekat base dan menemukan diamond lain di
        area terdekat }
    else if (basedistance(board_bot) == 2 and diamonds in game > 2) or
    (basedistance(board_bot) == 1 and bot's diamonds > 0):
        goal_position <- base of board_bot
        { Jika inventory penuh, kembali ke base }
    else if board_bot's diamonds == 5:
        goal_position <- base board_bot

    else if board_bot's diamonds > 3:
    { Jika ditemukan diamonds di area-area dekat bot }
        if closestdiamond(board_bot, board) is not empty:
        { Jika inventory masih 3, maka jika menemukan red diamond akan
        langsung jadi 5. Cari red diamond terdekat }
            if bot's diamond == 3 and exist red diamonds within 3
            cells:
                goal_position <- closest red diamonds
    { Jika kondisi di atas tidak terpenuhi, dan ada diamond lain dalam
    jarak 3 sel }
        else if closestdiamonddist(board_bot, board) <= 3:
            goal_position <- closestdiamond(board_bot, board)
```

```
        else:
{ Jika tidak, langsung ke base saja untuk mengamankan diamond yang
minimal 3 tadi }
            base <- base of board_bot
            goal_position <- base
{ Jika dekat base ternyata ada diamond lagi, ambil dulu saja sambil
perjalanan ke base }
            else if cekdiamondbase(board_bot, board):
                diamond_list <- diamondsaroundbase(board_bot, board)
                goal_position <- closestdiamondbase(board_bot,
diamond_list)
{ Mengamankan minimal 3 poin dulu ke base daripada keliling lebih jauh
lagi }
            else:
                base <- base of board_bot
                goal_position <- base

        { Jika diamond di inventory kurang dari sama dengan 3 }
        elif board_bot's diamonds <= 3:
{ Jika saat ini bot berada di sekitar base dan ada diamond sekitaran
base }
            if (cekdiamondbase(board_bot, board) and
botaroundbase(board_bot)) or (cekdiamondbase(board_bot, board) and
banyak diamond around base >= 3):
                diamond_list <- diamondsaroundbase(board_bot, board)
                goal_position <- closestdiamondbase(board_bot,
diamond_list)
{ Jika tidak ada, dan ternyata ada bot musuh dengan diamond lebih
banyak dari 3 dan lebih banyak dari kita, kejar }
            else if chaseBots(board_bot, board):
                increment chaseSteps
{ Jika tidak ada bot musuh atau diamond dalam jarak yang lebih dekat
dibanding jarak ke red button, ke red button }
            else if compareClosestDiamondToRedButton(board_bot, board):
                goal_position <- findRedButton(board).position
{ Jika tidak ada red button dalam jarak dekat, pergi ke diamond yang
jauh }
            else if closestreddiamond(board_bot, board) is not empty:
                if closestdiamond(board_bot, board) is not empty:
                    if closestreddiamonddist(board_bot, board) <
```



```
closestdiamonddist(board_bot, board):  
{ Memprioritaskan red diamond jika jaraknya lebih dekat }  
    goal_position <- closestreddiamond(board_bot,  
board)  
{ Jika tidak ambil saja yang biru asalkan lebih dekat }  
    else:  
        goal_position <- closestdiamond(board_bot, board)  
    else:  
        goal_position <- closestreddiamond(board_bot, board)  
    else:  
        goal_position <- closestdiamond(board_bot, board)  
{ Jika tidak ada juga, ambil saja diamond yang ada }  
  
{ Default goal jika position empty }  
    if goal_position is empty:  
        goal_position <- base of board_bot  
  
{ Utilisasi teleporter jika jarak menggunakan teleporter lebih dekat  
dibandingkan jarak tanpa teleporter }  
    if goal_position is base of board_bot:  
        goToBaseWithTeleporter(board_bot, board)  
  
{ Return langkah per langkah }  
    next_direction <- get_directions(  
        current_position of x,  
        current_position of y,  
        goal_position of x,  
        goal_position of y,  
    )  
  
->    next_direction
```

4.2. Struktur Data Program

Struktur data pada permainan Diamonds ini terdiri dari kelas-kelas yang terbagi dalam beberapa file. Terdapat dua folder yang digunakan dalam pengembangan bot yaitu, game dan logic.

4.2.1. Game

Folder game berisi pendefinisian model-model objek, posisi, peta, serta aksi yang valid dalam permainan. Berikut *file* yang ada dalam folder ini.

a. api.py

Tabel 6. *Tabel Struktur Data “api.py”*

api.py	
Atribut	<ul style="list-style-type: none"> ○ String url
Method	<ul style="list-style-type: none"> ○ def _get_url(self, endpoint: str) -> str ○ def _req(self, endpoint: str, method: str, body: dict) -> Response ○ def bots_get(self, bot_token: str) -> Optional[Bot] ○ def bots_register(self, name: str, email: str, password: str, team: str) -> Optional[Bot] ○ def boards_list(self) -> Optional[List[Board]] ○ def bots_join(self, bot_token: str, board_id: int) -> bool ○ def boards_get(self, board_id: str) -> Optional[Board] ○ def bots_move(self, bot_token: str, direction: str) -> Optional[Board] ○ def bots_recover(self, email: str, password: str) -> Optional[str] ○ def _return_response_and_status(self, response: Response) -> Tuple[Union[dict, List], int]

b. board_handler.py

Tabel 7. *Tabel Struktur Data “board_handler.py”*

board_handler.py	
Atribut	<ul style="list-style-type: none"> ○ Api api
Method	<ul style="list-style-type: none"> ○ def list_boards(self) -> List[Board] ○ def get_board(self, board_id: int) -> Board

c. bot_handler.py

Tabel 8. *Tabel Struktur Data* “bot_handler.py”

bot_handler.py	
Atribut	<ul style="list-style-type: none"> ○ Api api
Method	<ul style="list-style-type: none"> ○ def _get_direction(dx: int, dy: int) ○ def get_my_info(self, token: str) -> Bot ○ def join(self, token: str, board_id: int) -> bool ○ def move(self, token: str, board_id: int, dx: int, dy: int) -> Optional[Board] ○ def register(self, name: str, email: str, password: str, team: str) -> Optional[Bot] ○ def recover(self, email: str, password: str) -> Optional[str]

d. models.py

Tabel 9. *Tabel Struktur Data* “models.py”

Kelas Bot	
Atribut	<ul style="list-style-type: none"> ○ String name ○ String email ○ String id
Method	-
Kelas Position	
Atribut	<ul style="list-style-type: none"> ○ Int y ○ Int x
Method	-

Kelas Properties	
Atribut	<ul style="list-style-type: none"> Optional[int] points Optional[int] pair_id Optional[int] diamonds Optional[int] score Optional[int] name Optional[int] inventory_size Optional[int] can_tackle Optional[int] milliseconds_left Optional[int] time_joined Optional[int] base
Method	-
Kelas Game Object	
Atribut	<ul style="list-style-type: none"> Position position String type Optional[Properties] properties
Method	-
Kelas Config	
Atribut	<ul style="list-style-type: none"> Optional[float] generation_ratio Optional[float] min_ratio_for_generation Optional[float] red_ratio Optional[int] seconds Optional[int] pairs Optional[int] inventory_size Optional[bool] can_tackle

Method	-
Kelas Feature	
Atribut	<ul style="list-style-type: none"> ○ String name ○ Optional[Config] config
Method	-
Kelas Board	
Atribut	<ul style="list-style-type: none"> ○ Int id ○ Int width ○ Int height ○ List[Feature] features ○ Int minimum_delay_between_moves ○ Optional[List[GameObject]] game_objects
Method	<ul style="list-style-type: none"> ○ def bots(self) -> List[GameObject] ○ def diamonds(self) -> List[GameObject] ○ def get_bot(self, bot: Bot) -> Optional[GameObject] ○ def is_valid_move(self, current_position: Position, delta_x: int, delta_y: int) -> bool

e. util.py

Tabel 10. *Tabel Struktur Data “util.py”*

util.py	
Atribut	-
Method	<ul style="list-style-type: none"> ○ def clamp(n, smallest, largest) ○ def get_direction(current_x, current_y, dest_x, dest_y)

	<ul style="list-style-type: none"> ○ <code>def position_equals(a: Position, b: Position)</code>
--	--

4.2.2. Logic

Tabel 11. *Tabel Struktur Data "padibot.py"*

padibot.py	
Atribut	<ul style="list-style-type: none"> ○ <code>List[int] direction</code> ○ <code>Optional[Position] goal_position</code> ○ <code>Int current_direction</code> ○ <code>Int chaseSteps</code>
Method	<ul style="list-style-type: none"> ○ <code>def diamondsaroundbase(board_bot: GameObject, board: Board)</code> ○ <code>def botaroundbase(board_bot: GameObject)</code> ○ <code>def closestdiamondbase(board_bot: GameObject, diamonds: List[Position])</code> ○ <code>def cekdiamondbase(board_bot: GameObject, board: Board)</code> ○ <code>def closestdiamond(board_bot: GameObject, board: Board)</code> ○ <code>def closestdiamondddist(board_bot: GameObject, board: Board)</code> ○ <code>def basedistance(board_bot: GameObject)</code> ○ <code>def calculateDistanceToBots(board_bot: GameObject, enemy_bot: GameObject)</code> ○ <code>def findAllBots(board_bot: GameObject, board: Board)</code> ○ <code>def chaseBots(board_bot: GameObject, board: Board)</code> ○ <code>def closestreddiamond(board_bot: GameObject, board: Board)</code> ○ <code>def closestreddiamondddist(board_bot: GameObject, board: Board)</code> ○ <code>def findRedButton(self, board: Board)</code> ○ <code>def compareClosestDiamondToRedButton(board_bot: GameObject, board: Board)</code>

	<ul style="list-style-type: none"> ○ <code>def calculateDistanceRedButton(board_bot: GameObject, board: Board)</code> ○ <code>def findAllTeleporter(board_bot: GameObject, board: Board)</code> ○ <code>def goToBaseWithTeleporter(board_bot: GameObject, board: Board)</code> ○ <code>def get_directions(current_x, current_y, dest_x, dest_y)</code> ○ <code>def next_move(board_bot: GameObject, board: Board)</code>
--	--

4.3. Analisis dan Pengujian

Kami melakukan analisis dan pengujian terhadap Padibot (bot yang dibuat oleh kelompok kami) dengan bot referensi dari kelompok lain. Hasil dari pengujian dapat berubah dengan dinamis karena peta pada permainan bisa berubah-ubah setiap permainan dimulai kembali. Kami melakukan pengujian pada beberapa aksi yang dapat dilakukan oleh bot.

Kami juga menaruh bot kami pada posisi pertama dan terakhir, dan dari hasil pengujian ternyata urutan juga berpengaruh pada poin yang didapatkan. Poin bot kami pada urutan terakhir (bot random) memiliki total skor yang lebih besar daripada bot kami pada urutan pertama. Hal ini bisa terjadi karena waktu yang dimiliki oleh bot terakhir lebih banyak sehingga waktu untuk mentackle atau menghindari lawan juga lebih banyak dan berlian yang didapat juga bisa lebih banyak daripada saat bot berada di posisi pertama. Berikut hasil rekap poin pertandingan yang didapat:

Tabel 12. *Tabel Poin Pertandingan Padibot & Kawannya*

Bot	I	II	III	IV	V	Total Score
ilmupadi	19	19	18	27	21	104
sanss	17	15	24	17	18	91
tulas	19	12	27	12	16	86
random (ilmupadi)	23	17	22	24	20	106

4.3.1. Pengujian I

Tabel 13. *Tabel Hasil Pengujian I dan Analisisnya*

Gambar	Analisis
	<p>Saat Padibot (bot berada di tengah) jauh dari berlian dan lebih dekat ke arah <i>red button</i> maka bot akan bergerak ke <i>red button</i> untuk mengacak ulang lokasi berlian. Hal ini mengimplementasikan <i>greedy by Shortest Path to Current Position</i> sehingga aksi yang telah dilakukan optimum.</p>
	<p>Hasil dari pengujian pada permainan pertama.</p>

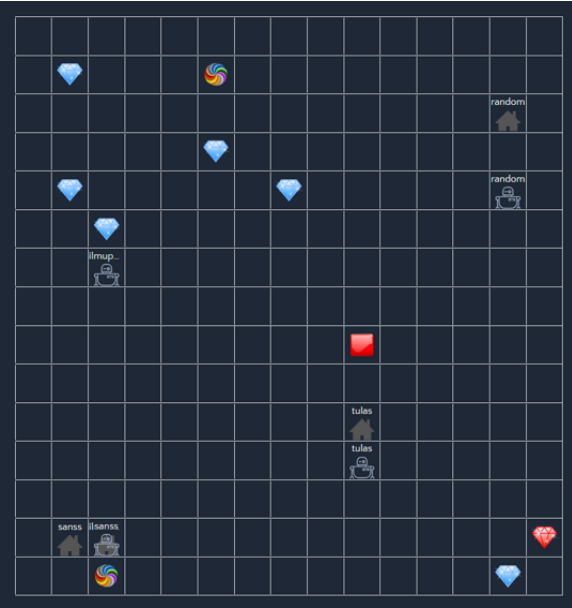
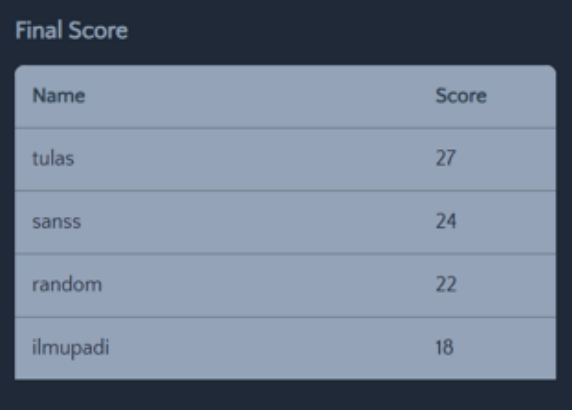
4.3.2. Pengujian II

Tabel 14. Tabel Hasil Pengujian II dan Analisisnya

Gambar	Analisis
	<p>Saat Padibot (bot berada di tengah) dan jarak ke teleporter dekat dengan base.</p> <p>Karena posisi bot lebih dekat ke teleporter dari pada jarak ke base maka bot akan memilih bergerak ke teleporter. Hal ini menerapkan <i>Greedy Closest to Base</i> sehingga aksi yang dilakukan optimum.</p>
	<p>Hasil dari pengujian pada permainan kedua.</p>

4.3.3. Pengujian III

Tabel 15. Tabel Hasil Pengujian III dan Analisisnya

Gambar	Analisis										
	<p>Saat base Padibot bersebelahan dengan base lawan (kiri bawah) rentan untuk berlian yang dimiliki dicuri oleh lawan saat hendak kembali ke base dengan membawa cukup banyak berlian. Sehingga usaha ke base akan berjalan sia sia saat mengimplementasikan menerapkan <i>Greedy Closest to Base</i> dan aksi berjalan tidak optimum.</p>										
 <table border="1"> <thead> <tr> <th>Name</th><th>Score</th></tr> </thead> <tbody> <tr> <td>tulas</td><td>27</td></tr> <tr> <td>sanss</td><td>24</td></tr> <tr> <td>random</td><td>22</td></tr> <tr> <td>ilmupadi</td><td>18</td></tr> </tbody> </table>	Name	Score	tulas	27	sanss	24	random	22	ilmupadi	18	<p>Hasil dari pengujian pada permainan ketiga.</p>
Name	Score										
tulas	27										
sanss	24										
random	22										
ilmupadi	18										

4.3.4. Pengujian IV

Tabel 16. Tabel Hasil Pengujian IV dan Analisisnya

Gambar	Analisis
	<p>Pada saat ingin mengambil berlian terdekat dan harus melewati teleporter yang berdekatan seperti kasus di samping, bot akan masuk ke teleporter dan bergerak ke teleporter lainnya sehingga waktu akan terbuang sia-sia. Maka penerapan <i>greedy by Shortest Path to Current</i> pada saat dilakukan dilakukan tidak optimum.</p>
	<p>Hasil dari pengujian pada permainan keempat.</p>

4.3.5. Pengujian V

Tabel 17. Tabel Hasil Pengujian V dan Analisisnya

Gambar	Analisis
 	<p>Pada saat banyak berlian di sekitar <i>base</i>, Padibot (tengah kiri) akan mengambil bot yang ada di sekitar <i>base</i> terlebih dahulu. Pada saat ini bot menerapkan <i>Greedy Closest to Base</i> dan aksi berjalan optimum.</p>
	<p>Hasil dari pengujian pada permainan kelima.</p>

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari tugas besar IF2211 Strategi Algoritma ini, kami telah berhasil membuat bot permainan Diamonds menggunakan strategi *greedy*, yaitu dengan mencari langkah yang menghasilkan poin sebanyak-banyaknya. Strategi yang digunakan mengimplementasikan kombinasi dari beberapa strategi *greedy* yang memiliki prioritasnya masing-masing, dimulai dari mencari *diamond* dengan jarak terdekat ke *base* dan *diamond* terdekat ke bot, sehingga terciptalah sebuah bot dengan strategi *greedy* versi Ilmu Padi.

5.2. Saran

Adapun beberapa saran pengembangan untuk tugas besar ini adalah:

1. Menghindari penyelesaian tugas besar secara tergesa-gesa mendekati *deadline* agar bot yang dihasilkan dapat dikembangkan dengan lebih baik dan sempurna.
2. Strategi yang diterapkan perlu dioptimalkan lebih lanjut, baik dari segi format kode maupun efektivitas kode, untuk memastikan bahwa bot yang dibuat dapat berperforma lebih baik dalam berbagai kondisi permainan, seperti utilisasi objek pada permainan secara maksimal yang perlu lebih diperhatikan lagi, sambil meminimalisir kompleksitas logika bot.
3. Mencari tahu lebih lanjut mengenai cara kerja dan struktur dalam permainan, misalnya mengenai cara kerja *controller* dan *requests*, dan bertanding lebih sering dengan kawan untuk mencari kekurangan dan memperbaiki algoritma bot.

DAFTAR PUSTAKA

- [1] *geeksforgeeks.com*, “Greedy Algorithm”. Diakses Jumat, 8 Maret 2024. Tersedia pada: <https://www.geeksforgeeks.org/greedy-algorithms/>
- [2] *github.com*, “diamonds2”. Terakhir diakses Sabtu, 9 Maret 2024. Tersedia pada: <https://github.com/Etimo/diamonds2>
- [3] *informatika.stei.itb.ac.id*, “Algoritma Greedy Bag 1”. Diakses Senin, 4 Maret 2024. Tersedia pada: [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)
- [4] *informatika.stei.itb.ac.id*, “Algoritma Greedy Bag 2”. Diakses Senin, 4 Maret 2024. Tersedia pada: [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)
- [5] *informatika.stei.itb.ac.id*, “Algoritma Greedy Bag 3”. Diakses Senin, 4 Maret 2024. Tersedia pada: [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)

LAMPIRAN

A. Repository Github

Link *repository* Github: https://github.com/bagassambega/Tubes1_ilmu-padi

B. Youtube Video

Link *video* YouTube: <https://youtu.be/TY8sodSkbK8>

C. Naskah Video Youtube

#Di Sebuah Pedesaan

{ Ada seorang pengangguran alumni computer science di sebuah coffee shop yang dekat dengan sawah }

Nabila: “ Ehh Bagas, Apa kabar? kamu udah lulus dari **computer science** NTU yaa??”

Bagas : “ Iya nihh, sekarang nganggur :(, kadang nyambi jadi petani tapi sekarang sawah banjir beras pada rusak” {sambil mainin beras rusak}

Nabila: “ Duhh memang sihh, curah hujan sekarang lagi tinggi banget, hmm gimana kalau kamu coba main game ini” {mencoba menyemangati}

Bagas: “oke aku coba”

{ Beberapa saat kemudian }

Bagas: “ih gampang banget ini game”

Shika: “pantesan kamu pengangguran kamu sombong gitu, kamu harus selalu menerapkan **ilmu padi**, coba deh bisa gak bikin **bot** buat game ini”

Bagas: “bot doang mah gampang”

Shika: “coba nanti tanding sama punyaku ya, kalau kalah berarti cupu”

{mencoba tanding antar bot di **game diamonds**}

Shika: “loh ternyata kalah, kenapa ya? ”

Shika: “kamu pake algoritma apa sih gass? Lulusan **computer science** NTU memang ga kaleng-kaleng ”

Bagas: “pake **algoritma greedy** dongg!”

Nabila: “shik, **algoritma greedy** itu apa ya? ”

Shika: {Menjelaskan garis besar **algoritma greedy**}

Nabila: “oalahh, si Bagas nerapinnya gimana ya kok bisa dia menang?”

Shika: ”gatau ya coba kita tanya dia”

{Bagas menjelaskan **algoritma greedy by Closest to Base** yang dibuat}

{Nabila menjelaskan **Analogi Greedy by Closest to Base** -> ada yang lagi bagi2 uang cari yang dekat dulu karena pencuri banyak -> buru buru simpen}

Nabila: “pantesan pengangguran orang kamu **greedy!**”