

# LAPORAN TUGAS BESAR 1 IF2211

## STRATEGI ALGORITMA

*Pemanfaatan Algoritma Greedy  
dalam Pembuatan Bot Permainan Diamonds*



**Dosen Pengampu** : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.  
Dr. Ir. Rinaldi Munir, M.T.  
**Asisten pembimbing** : Bernardus Willson

**Disusun oleh:**

**Kelompok 39 - Jadi Mesin**

<b>Panji Sri Kuncara Wisma</b>	<b>(13522028)</b>
<b>Ahmad Hasan Albana</b>	<b>(13522041)</b>
<b>Amalia Putri</b>	<b>(13522042)</b>

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

**2024**

## **KATA PENGANTAR**

Puji syukur kita panjatkan ke hadirat Allah SWT, karena atas rahmat dan karunia-Nya, kami dapat menyelesaikan makalah ini dengan judul "Pemanfaatan Algoritma Greedy dalam Pembuatan Bot Permainan Diamonds". Makalah ini disusun sebagai salah satu tugas mata kuliah Strategi Algoritma, dengan fokus pada penerapan konsep Algoritma Greedy dengan keenam elemen-elemennya termasuk suatu jenis metode Algoritma Greedy yang dipakai.

Penyusunan makalah ini tidak lepas dari bantuan berbagai pihak. Kami mengucapkan terima kasih kepada dosen beserta asisten pembimbing mata kuliah Strategi Algoritma yang telah memberikan bimbingan dan panduan selama proses pembuatan makalah ini.

Semoga makalah ini dapat memberikan kontribusi positif dalam pemahaman dan pengembangan suatu aplikasi dari konsep Algoritma Greedy. Akhir kata, kami menyampaikan permohonan maaf atas segala keterbatasan dalam makalah ini, dan kami menerima dengan terbuka segala kritik dan saran yang bersifat membangun.

Bandung, 5 Maret 2024,

Panji Sri Kuncara Wisma	(13522028)
Ahmad Hasan Albana	(13522041)
Amalia Putri	(13522042)

## **DAFTAR ISI**

<b>KATA PENGANTAR.....</b>	<b>1</b>
<b>DAFTAR ISI.....</b>	<b>2</b>
<b>DAFTAR PEMBAGIAN TUGAS.....</b>	<b>3</b>
<b>BAB I.....</b>	<b>4</b>
1.1. Abstraksi.....	4
<b>BAB II.....</b>	<b>7</b>
2.1. Algoritma Greedy.....	7
2.2. Integer Knapsack Problem.....	8
2.3. Cara Kerja Permainan Diamonds.....	8
2.4. Implementasi Algoritma Greedy.....	9
<b>BAB III.....</b>	<b>12</b>
3.1 Langkah-Langkah Pemecahan Masalah.....	12
<b>BAB IV.....</b>	<b>17</b>
4.1 Implementasi Algoritma Greedy.....	17
4.2 Penjelasan Struktur Data pada Program.....	25
4.3 Analisis Desain Solusi Algoritma.....	25
4.4 Kasus Pengujian.....	27
<b>BAB V.....</b>	<b>28</b>
5.1. Kesimpulan.....	28
5.2. Saran.....	28
<b>DAFTAR PUSTAKA.....</b>	<b>29</b>
<b>LAMPIRAN.....</b>	<b>30</b>
<b>Repository.....</b>	<b>30</b>
<b>Youtube.....</b>	<b>30</b>

## DAFTAR PEMBAGIAN TUGAS

	KEGIATAN	PIC
BOT (LOGIC)	Menggerakkan bot ke diamond dengan jarak terdekat	13522028 13522041
	Menggerakkan bot ke diamond dengan keuntungan terbesar	13522042
	Menggerakkan bot ke base kalau sisa step sama dengan sisa waktu	13522042
	Menggerakkan bot ke diamond dengan densitas (keuntungan/jarak)	13522042
	Menggerakkan bot berdasarkan kondisi jarak ke diamond terhadap musuh	13522028
	Menggerakkan bot ke red diamond	13522041
	Menggerakkan bot ke teleporter	13522028 13522041
LAPORAN	BAB I: Deskripsi Tugas	13522041 13522042 13522028
	BAB II: Landasan Teori	13522042
	BAB III: Aplikasi Strategi Greedy	13522028
	BAB IV: Implementasi dan Pengujian	13522041
	BAB V: Kesimpulan dan Saran	13522041 13522042 13522028
BONUS	Youtube Video	13522041 13522042 13522028

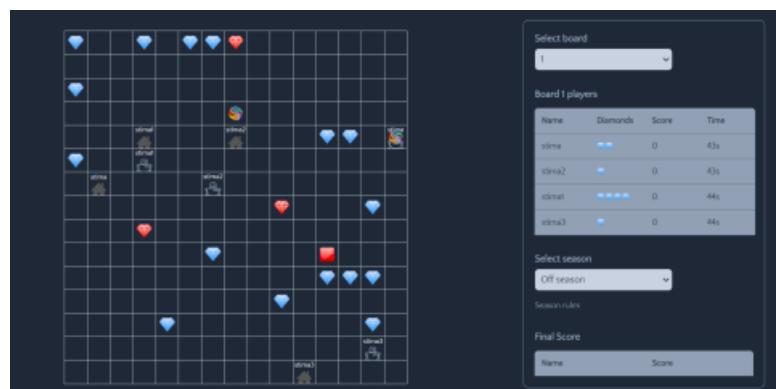
**Tabel 1.** Pembagian tugas dalam pembuatan Tugas Besar

# BAB I

## DESKRIPSI TUGAS

### 1.1. Abstraksi

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.



Gambar 1. Permainan Diamonds

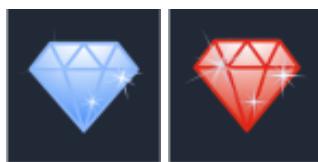
Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan strategi greedy dalam membuat bot ini. Program permainan Diamonds terdiri atas

- 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

Terdapat pula komponen-komponen dari permainan Diamonds antara lain, yakni sebagai berikut.

### 1) Diamonds



**Gambar 2.** Diamond Biru dan Merah

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



**Gambar 3.** Red/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.

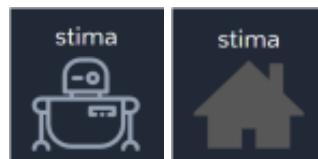
### 3) Teleporters



Gambar 4. 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



Gambar 5. 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. Posisi *Base* tidak akan berubah sampai akhir game.

### 5) Inventory

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

Gambar 6. 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.

## **BAB II**

### **LANDASAN TEORI**

#### **2.1. Algoritma Greedy**

Algoritma Greedy merupakan metode yang paling populer dan sederhana untuk memecahkan persoalan optimasi, serta ada dua macam persoalan optimasi, yakni maksimasi dan minimasi. Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah (step by step) sedemikian sehingga pada setiap langkah itu mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Berikut elemen-elemen yang terdapat dalam Algoritma Greedy yang harus ditentukan dan dipertimbangkan.

- 1) Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
- 2) Himpunan solusi, S : berisi kandidat yang sudah dipilih
- 3) Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
- 4) Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
- 5) Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
- 6) Fungsi obyektif : memaksimumkan atau meminimumkan

Namun, optimum global belum tentu merupakan solusi optimum (terbaik), bisa jadi merupakan solusi sub-optimum atau pseudo-optimum. Hal ini dikarenakan Algoritma greedy tidak beroperasi secara menyeluruh terhadap semua kemungkinan solusi yang ada dan terdapat beberapa fungsi seleksi yang berbeda sehingga harus memilih fungsi yang tepat jika ingin algoritma menghasilkan solusi optimal.

## 2.2. Integer Knapsack Problem

Terdapat penggunaan konsep Integer Knapsack Problem dalam implementasi Algoritma Greedy. Hal ini membantu dalam mengoptimasi solusi, namun karena implementasi Algoritma Greedy juga menerapkan teori heuristik sehingga suatu langkah yang sudah dilakukan tidak dapat dilakukan *backtracking*. Dengan konsep ini, setiap objek memiliki properti beban yang dapat berupa jarak, bobot, dan lainnya. Lalu, memiliki properti keuntungan dan juga terdapat rasio antara keuntungan dan beban, yakni densitas. Oleh karena itu, Algoritma Greedy dapat dibagi berdasarkan 3 properti:

- 1) *Greedy by profit* (keuntungan), yakni pada setiap langkah, pilih objek yang mempunyai keuntungan terbesar dan strategi ini mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan terlebih dahulu.
- 2) *Greedy by weight* (beban), yakni pada setiap langkah, pilih objek yang mempunyai beban tercukup dari kapasitas maksimum dengan mencoba memaksimumkan keuntungan dengan memasukkan sebanyak mungkin objek ke dalam knapsack.
- 3) *Greedy by density* (densitas), yakni pada setiap langkah, knapsack diisi dengan objek yang mempunyai *profit/weight* terbesar dengan mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar.

## 2.3. Cara Kerja Permainan Diamonds

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

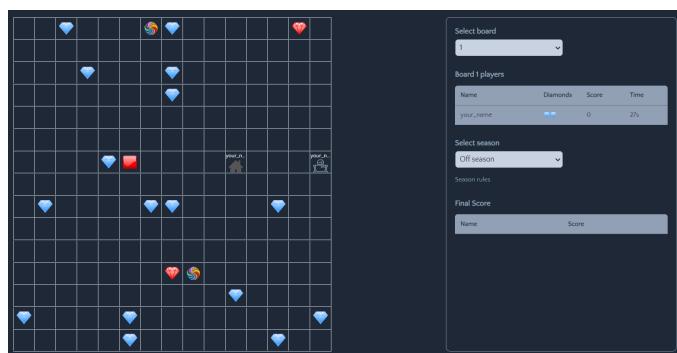
- 1) Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
- 2) Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
- 3) Objektif utama bot adalah 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.

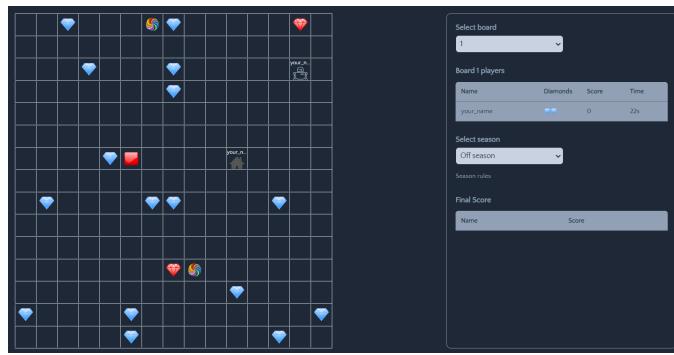
- 4) Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
- 5) Apabila bot menuju ke posisi home base, score bot akan bertambah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

#### 2.4. Implementasi Algoritma Greedy

Berdasarkan cara bermain dan kondisi permainan, dapat diketahui bahwa bot akan melangkah dengan kecepatan 1 langkah/detik dan juga terdapat 2 jenis diamond, yakni red diamond yang berbobot 2 poin dan blue diamond yang berbobot 1 poin. Artinya, objek yang menjadi properti beban adalah langkah/waktu (kecepatan) sedangkan objek yang menjadi properti keuntungan adalah diamond.



Gambar 7.1 Implementasi Algoritma Greedy Bot-Diamond



**Gambar 7.2** Implementasi Algoritma Greedy Bot-Diamond

Dari posisi awal pada **Gambar 7.1** dan posisi akhir pada **Gambar 7.2**, diperlihatkan bahwa terdapat 2 perbandingan dari diamond yang terdekat, yaitu ada red diamond pada posisi (13, 0) dan blue diamond (12,8). Kondisi ini, di ilustrasikan pada tabel berikut.

Properti objek				<i>Greedy by</i>			Solusi Optimal
i	w <sub>i</sub>	p <sub>i</sub>	p <sub>i</sub> / w <sub>i</sub>	profit	weight	density	
1	7	2	0,29	1	0	1	1
2	4	1	0,25	0	1	0	0
Total bobot				2	1	2	2
Total keuntungan				2	1	2	2

**Tabel 1.** Implementasi Algoritma Greedy Bot-Diamond

Berdasarkan hasil tersebut, penulis memutuskan untuk mengimplementasikan konsep Algoritma Greedy *by density* untuk mendekati kasus optimal. Selain itu, ini detail dari keenam elemen Algoritma Greedy sebagai bahan pertimbangan langkah yang dilakukan oleh bot jika mendapati ketiga komponen tersebut.

- 1) Himpunan kandidat, C : Semua langkah yang mungkin dilakukan oleh bot untuk mendekati dan mengambil diamond. Ini termasuk pergerakan ke berbagai lokasi di mana diamond tersedia.
- 2) Himpunan solusi, S : Langkah dari Himpunan Kandidat yang menghasilkan pengambilan jumlah diamond terbanyak, termasuk mempertimbangkan

penggunaan Red Button untuk memaksimalkan jumlah diamond yang dapat diambil.

- 3) Fungsi solusi: Fungsi yang mengevaluasi apakah kondisi akhir telah tercapai, bisa jadi berupa bot telah mengumpulkan jumlah diamond tertentu atau waktu permainan telah habis. Tujuan akhir adalah untuk memaksimalkan jumlah diamond yang dikumpulkan sebelum kembali ke base.
- 4) Fungsi seleksi (selection function): Memilih langkah selanjutnya berdasarkan kriteria jumlah diamond terbanyak yang bisa diambil dalam satu langkah, memperhitungkan posisi diamond, Red Button, dan Teleporters.
- 5) Fungsi kelayakan (feasible): Semua langkah dianggap layak selama memungkinkan bot untuk bergerak ke lokasi diamond atau Red Button tanpa atau kehilangan diamond karena inventory penuh.
- 6) Fungsi obyektif : Memaksimalkan jumlah diamond yang dikumpulkan oleh bot. Strategi ini memprioritaskan pengambilan diamond merah dan menggunakan Red Button secara strategis.

## **BAB III**

### **APLIKASI STRATEGI GREEDY**

#### **3.1 Langkah-Langkah Pemecahan Masalah**

Berikut adalah elemen-elemen dari algoritma Greedy yang digunakan dalam pembuatan *bot* permainan Diamonds :

- Himpunan kandidat yang berpotensi dipilih untuk setiap pergerakan adalah objek-objek yang ada pada papan yaitu diamond biru, diamond merah, bot musuh, base, portal teleportasi, dan tombol merah.
- Himpunan solusi adalah himpunan diamond yang bot terdekatnya adalah bot sendiri, teleporter terdekat terhadap bot sendiri, redbutton, dan base.
- Fungsi solusi yang digunakan adalah fungsi untuk mengecek apakah total diamond yang berada di dalam *inventory* sudah memenuhi kapasitas maksimal atau belum.
- Fungsi seleksi : memilih diamond berdasarkan *density* yaitu diamond dengan jarak terdekat dan poin tertinggi. Nilai *density* didapat dengan membagi poin untuk setiap diamond dengan jarak dari bot utama. Jarak yang ditempuh melalui portal teleportasi juga menjadi pertimbangan dalam perhitungan *density*.
- Fungsi kelayakan : Memeriksa apakah kandidat diamond sudah pasti bisa diambil oleh bot dengan dua pertimbangan. Pertimbangan pertama adalah nilai *density* dari bot utama terhadap target diamond harus lebih besar dari nilai *density* terbesar yang mungkin dari bot musuh terhadap target yang dimaksud. Apabila kondisi tersebut tidak terpenuhi, untuk sementara waktu bot akan diarahkan ke tombol merah. Pertimbangan kedua adalah apakah waktu perjalanan mengambil diamond target dan kembali ke base masih cukup. Apabila jumlah langkah yang perlu diambil untuk mengambil diamond dan kembali ke base lebih besar dari waktu yang tersisa maka diamond tersebut tidak layak untuk diambil dan bot diharuskan untuk kembali ke base.
- Fungsi objektif: Jumlah poin yang didapat maksimal.

Selama proses perancangan algoritma greedy, ada beberapa alternatif yang menjadi pertimbangan. Alternatif solusi berikut dirancang dengan asumsi list diamond yang menjadi target sudah disaring menggunakan fungsi kelayakan dan sudah pasti dapat diambil. Berikut adalah uraiannya :

- a. Melakukan pemilihan diamond berdasarkan jarak terdekat

Ide utama dari solusi ini adalah dengan mengambil diamond dengan jarak terdekat. Jarak yang ditempuh melalui jalur teleportasi juga menjadi pertimbangan pada solusi ini. Tujuan dari pengambilan diamond dengan jarak terdekat adalah untuk meminimumkan jumlah langkah yang diambil oleh bot.

- b. Melakukan pemilihan diamond berdasarkan poin terbesar

Ide utama dari solusi ini adalah mengambil diamond dengan poin terbesar terlebih dahulu.

- c. Melakukan pemilihan diamond berdasarkan *density* terbesar dengan jarak minimum

Ide utama dari solusi ini adalah dengan mengambil diamond yang memiliki poin terbesar dengan jarak terdekat atau memiliki nilai poin dibagi jarak terbesar. Solusi ini juga mempertimbangkan jarak yang dilalui melalui jalur teleportasi. Jika ada diamond dengan *density* yang sama, maka akan dipilih diamond dengan nilai jarak yang minimum.

- d. Melakukan pemilihan diamond berdasarkan persebarannya pada suatu wilayah

dan mengambil diamond-diamond terdekat setelah sampai di wilayah tersebut  
Ide utama dari solusi ini adalah dengan membagi papan menjadi empat atau lebih kuadran dan menentukan jumlah diamond pada masing-masing kuadran. Setelah mengetahui informasi tersebut, bot akan diarahkan ke kuadran dengan jumlah diamond terbanyak dan mulai mengumpulkan poin di kuadran tersebut.

Analisis kelebihan dan kekurangan juga dilakukan terhadap solusi-solusi alternatif diatas yaitu sebagai berikut:

- a. Melakukan pemilihan diamond berdasarkan jarak terdekat

Kelebihan dari solusi ini adalah bahwa jumlah langkah yang bot ambil hampir atau bisa minimum. Kekurangan dari solusi ini adalah bot akan mengambil diamond apapun selama jarak diamond tersebut dekat. Dengan jarak yang

sama bisa saja diamond dengan poin satu akan menjadi pilihan ketimbang diamond dengan poin dua. Pada beberapa kasus hal tersebut cukup merugikan. Kompleksitas algoritma ini adalah  $O(n)$  untuk Best Case dan Worst Case, dengan  $n$  adalah jumlah diamond yang berada pada *board*. Hal tersebut dikarenakan solusi ini memerlukan pengecekan dan pembandingan satu per satu diamond untuk mendapatkan diamond terdekat.

b. Melakukan pemilihan diamond berdasarkan poin terbesar

Kelebihan dari solusi ini adalah bot berpotensi besar mendapatkan poin terbesar untuk setiap pengambilan diamond. Akan tetapi ada beberapa kekurangan yang berisiko. Kekurangan pertama adalah pada situasi jarak diamond dengan poin terbesar terlalu jauh dari bot tapi jarak pada jalan menuju tujuan terdapat diamond yang sebenarnya masih bisa diambil. Hal tersebut tentunya bisa sangat merugikan. Kompleksitas algoritma ini adalah  $O(1)$  untuk Best Case dan  $O(n)$  untuk Worst Case, dengan  $n$  adalah jumlah diamond yang berada pada *board*. Hal tersebut dikarenakan solusi ini memerlukan pengecekan dan pembandingan satu per satu diamond untuk mencari diamond dengan points 2 dan pencarian dapat langsung dihentikan jika sudah ditemukan.

c. Melakukan pemilihan diamond berdasarkan *density* terbesar dengan jarak minimum

Kelebihan dari solusi ini adalah bot bisa mendapatkan diamond yang layak dengan jarak terdekat dan poin terbesar. Kekurangan yang mungkin dari solusi ini adalah jika bot berada pada lokasi dengan persebaran diamond yang cukup renggang tapi *density* diamond bot pada lokasi tersebut lebih besar daripada lokasi lain dengan persebaran yang lebih padat. Akan tetapi, sebenarnya kekurangan tersebut tidak terlalu berisiko jika dibandingkan dengan alternatif solusi yang lain. Kompleksitas algoritma ini adalah  $O(n)$  untuk Best Case dan Worst Case, dengan  $n$  adalah jumlah diamond yang berada pada *board*. Hal tersebut dikarenakan solusi ini memerlukan pengecekan dan pembandingan

satu per satu diamond untuk mendapatkan diamond dengan *density* terbesar dan jarak terdekat.

- d. Melakukan pemilihan diamond berdasarkan persebarannya pada suatu wilayah
- Kelebihan dari solusi ini adalah bahwa lokasi yang dikunjungi oleh bot akan memiliki diamond yang melimpah. Bot bisa dengan lebih cepat mengumpulkan mengumpulkan 5 poin di area tersebut dan segera kembali ke base. Akan tetapi, solusi ini tidak diambil karena memiliki kekurangan yang cukup berisiko. Kekurangan yang dimaksud adalah jika jarak base ke lokasi diamond dengan persebaran diamond terlalu jauh. Akibatnya, bot memerlukan waktu yang lama untuk bergerak bolak balik dari base ke lokasi tujuan. Ada kemungkinan juga bot akan mengabaikan diamond-diamond yang lebih dekat dengan base dan lebih mementingkan diamond dengan jarak yang lebih jauh. Hal tersebut juga bisa sangat merugikan untuk banyak kasus. Kompleksitas algoritma ini adalah  $O(n \log n)$  untuk Best Case dan Worst Case, dengan  $n$  adalah jumlah diamond yang berada pada *board*. Hal tersebut dikarenakan solusi ini memerlukan pengurutan diamond berdasarkan posisinya pada *board* agar diamond dapat dengan mudah dikelompokkan untuk diamond yang saling berdekatan.

Solusi-solusi alternatif diatas memiliki kelebihan dan kekurangan masing-masing. Adapun kompleksitas solusi diatas dihitung diluar dari kompleksitas algoritma fungsi kelayakan. Setiap solusi akan lebih unggul dibandingkan dengan solusi yang lain pada situasi yang tepat. Akan tetapi, semua solusi tersebut memiliki satu kesamaan, yaitu apabila lokasi base dan diamond terlalu jauh kemungkinan solusi terburuk akan tetap terjadi. Oleh karena itu, hal tersebut tidak akan dijadikan pertimbangan.

Setelah melakukan beberapa pengamatan terhadap perilaku permainan, hampir seluruh kasus memiliki persebaran diamond yang hampir merata. Pada kasus tersebut alternatif solusi a dan c jauh lebih diunggulkan dibandingkan yang lain. Melihat dari risiko, alternatif a dan c juga relatif sama dan jauh lebih aman

dibandingkan solusi b dan d. Oleh karena itu alternatif solusi b dan d tereliminasi dari kandidat alternatif solusi.

Dari kandidat alternatif solusi yang tersisa yaitu a dan c. Solusi c menjadi strategi greedy yang dipilih. Alasan utamanya adalah karena solusi c sudah mencakup pencarian diamond dengan jarak terdekat yang menjadi ide utama dari solusi a. Pada beberapa percobaan, solusi c hampir selalu muncul sebagai pemenang pertandingan. Selain itu, solusi a tidak dipilih juga karena solusi tersebut akan mengabaikan informasi perbedaan poin diamond yang cukup krusial pada permainan ini. Oleh karena itu , pada permainan Diamonds, bot utama akan memanfaatkan strategi greedy memilih diamond berdasarkan *density* terbesar dengan jarak minimum sebagai solusi terpilih.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Algoritma Greedy

Berikut adalah implementasi program utama kami dalam bentuk *pseudocode*.

```
{PROGRAM GAME LOGIC}

from typing import Optional
from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
from ..util import get_direction

class SuperBot(BaseLogic):
    def __init__(self):
        self.directions ← [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] ← None
        self.current_direction ← 0

    def next_move(self, board_bot: GameObject, board: Board):
        //Fungsi clamp akan mengembalikan nilai n dengan rentang nilai smallest sampai ke largest. Jika n berada diluar rentang, maka mengembalikan batas dari rentang tersebut.
        def clamp(n, smallest, largest):
            → max(smallest, min(n, largest))

        // Fungsi position_equals akan me return true jika lokasi a dan lokasi b sama
        def position_equals(a: Position, b: Position):
            → a.x = b.x and a.y = b.y

        //Fungsi get_direction_Adv adalah modifikasi dari fungsi get_direction agar robot bisa menghindari lokasi lokasi pada avoidList
```

```

def get_direction_Adv(current_x: int, current_y: int, dest_x: int, dest_y: int, avoidList):
    listBaru ← [(a.x, a.y) for a in avoidList]
    delta_x ← clamp(dest_x - current_x, -1, 1)
    delta_y ← clamp(dest_y - current_y, -1, 1)
    if delta_x ≠ 0:

        isBlocked ← False
        if (delta_y ≠ 0 and dest_x-delta_x = current_x):
            for i in range(current_y, dest_y+delta_y, delta_y):
                if ((current_x + delta_x, i) in listBaru):
                    isBlocked ← True

        if (isBlocked or (current_x + delta_x, current_y) in listBaru):
            delta_x ← 0
            if delta_y = 0:
                if (current_y ≠ 0 and not((current_x, current_y-1) in listBaru)):
                    delta_x, delta_y ← 0, -1
                else:
                    delta_x, delta_y ← 0, 1
            else:
                delta_y ← 0
            if delta_x = 0:
                if ((current_x, current_y+delta_y) in listBaru):
                    if (current_x ≠ 0 and not((current_x-1, current_y) in listBaru)):
                        delta_x, delta_y ← -1, 0
                    else:
                        delta_x, delta_y ← 1, 0

    → delta_x, delta_y

```

```

props ← board_bot.properties

// Membuat array berisi objek objek yang ada pada game
tombol_merah ← [d for d in board.game_objects if d.type =
"DiamondButtonGameObject"]
teleportasi ← [d for d in board.game_objects if d.type =
"TeleportGameObject"]
current_position ← board_bot.position

arr_bot ← board.bots
list_diamonds ← board.diamonds

arr_1 ← []
arr_1_teleport ← []

// Melakukan filter agar mengeluarkan bot kita sendiri dari array
filtered_bots ← [bot for bot in arr_bot if not(bot.position.x =
current_position.x and bot.position.y = current_position.y)]

//Melakukan pencarian robot terdekat untuk setiap diamonds
for diamond in list_diamonds:
    list_robot_terdekat ← sorted(filtered_bots, key=λ bots:
abs(bots.position.x - diamond.position.x) + abs(bots.position.y -
diamond.position.y))

// Melakukan pencarian diamond, robot terdekat dari diamond tersebut baik
dari jalur biasa atau teleportasi, robot terdekat, dan lokasi teleportasi terdekat.
if filtered_bots:
    arr_1.append((diamond, list_robot_terdekat[0]))

```

```

        teleport_terdekat ← sorted(teleportasi, key←lambda teleport:
abs(teleport.position.x - diamond.position.x) + abs(teleport.position.y - diamond.position.y))

        list_robot_terdekat_teleportasi ← sorted(filtered_bots, key←lambda
bots: (abs(bots.position.x - teleport_terdekat[1].position.x) + abs(bots.position.y - teleport_terdekat[1].position.y)))

    if filtered_bots:
        arr_1_teleport.append((diamond, list_robot_terdekat_teleportasi[0]))

    arr_2 ← []
    arr_2_teleportasi ← []

//Mencari selisih jarak kita dengan diamond target dengan jarak musuh terdekat
dengan diamond yang dimaksud melalui jalur darat.

for elemen in arr_1:
    // ini buat jarak biasa
    distance_to_us ← abs(elemen[0].position.x - current_position.x) +
abs(elemen[0].position.y - current_position.y)

        selisih ← distance_to_us - (abs(elemen[1].position.x - elemen[0].position.x) + abs(elemen[1].position.y - elemen[0].position.y))

    arr_2.append((elemen[0], selisih, distance_to_us, 1))

//Mencari selisih jarak kita dengan diamond target dengan jarak musuh terdekat
dengan diamond yang dimaksud melalui jalur teleportasi

for elemen in arr_1_teleport:
    // ini buat jarak teleportasi
    teleport_terdekat ← sorted(teleportasi, key←lambda teleport:
(abs(teleport.position.x - elemen[0].position.x) + abs(teleport.position.y - elemen[0].position.y)))

```

```

        distance_to_us_teleportasi ← ((abs(element[0].position.x - teleport_terdekat[0].position.x) + abs(element[0].position.y - teleport_terdekat[0].position.y)) + (abs(teleport_terdekat[1].position.x - current_position.x) + abs(teleport_terdekat[1].position.y - current_position.y)))

        selisih_teleportasi ← distance_to_us_teleportasi - ((abs(element[0].position.x - teleport_terdekat[0].position.x) + abs(element[0].position.y - teleport_terdekat[0].position.y)) + (abs(teleport_terdekat[1].position.x - element[1].position.x) + abs(teleport_terdekat[1].position.y - element[1].position.y)))

        arr_2_teleportasi.append((element[0], selisih_teleportasi, distance_to_us_teleportasi, 2))

//Melakukan sort untuk mendapatkan jarak diamond terdekat melalui jalur biasa dan teleportasi .

arr_2_teleportasi ← sorted(arr_2_teleportasi, key=λ elem: elem[2])
arr_2 ← sorted(arr_2, key=λ elem: elem[2])

//Melakukan penggabungan hasil sort
arr_2 ← arr_2_teleportasi + arr_2

filtered_arr_2 ← []
filtered_arr_2 ← [elem for elem in arr_2 if (elem[1] < 0 and elem[2] ≠ 0)]

selected_goal ← tombol_merah[0]

// Ini artinya ada diamond yang layak untuk diambil maka bot akan di arahkan ke lokasi dengan density terbesar
if len(filtered_arr_2) > 0 and filtered_arr_2 ≠ []:
    // Ini mencari diamond dengan density terbesar
    min_distance_elem ← max(filtered_arr_2, key=λ elem: elem[0].properties.points / elem[2])

```

```

if min_distance_elem[3] = 1:
    selected_goal ← min_distance_elem[0]
    if not(position_equals(selected_goal.position, current_position)):
        selected_goal ← min_distance_elem[0]
    else:
        selected_goal ← tombol_merah[0]
    else:
        lokasi_teleport_cik ← sorted(teleportasi, key=λ teleport:
            (abs(teleport.position.x - min_distance_elem[0].position.x) +
            abs(teleport.position.y - min_distance_elem[0].position.y)))

        selected_goal ← lokasi_teleport_cik[1]
        if not(position_equals(selected_goal.position, current_position)):
            selected_goal ← lokasi_teleport_cik[1]
        else:
            selected_goal ← tombol_merah[0]

//Kalau misalkan gak ada diamond yang layak untuk diambil bot akan bergerak
ke tombol merah
else:
    selected_goal ← tombol_merah[0]
//Menghitung langkah ke base dan waktu tersisa
    steps_to_base ← abs(current_position.x - props.base.x) +
    abs(current_position.y - props.base.y)
    time_left ← int(board_bot.properties.milliseconds_left / 1000)

//Kalau misalkan poin sama dengan ukuran inventory atau langkah ke base sama
dengan sisa waktu atau kalau udah punya inventory - 1 poin dan target diamond
selanjutnya berpoin 2 maka bot harus kembali ke base
if (props.diamonds = props.inventory_size) or (steps_to_base = time_left)
or      (selected_goal.type = "DiamondGameObject" and

```

```

selected_goal.properties.points      = 2      and      props.diamonds      =
props.inventory_size-1):
base ← board_bot.properties.base
sort_base_to_teleport ← sorted(teleportasi, key=λ teleport:
(abs(teleport.position.x - base.x) + abs(teleport.position.y - base.y)))
jarak_base_biasa ← abs(base.x - current_position.x) + abs(base.y -
current_position.y)
jarak_base_teleportasi ← ((abs(base.x - sort_base_to_teleport[0].position.x) +
abs(base.y - sort_base_to_teleport[0].position.y)) +
abs(current_position.x - sort_base_to_teleport[1].position.x) +
abs(current_position.y - sort_base_to_teleport[1].position.y)))

// kalau jarak base biasa kurang dari jarak biasa teleportasi maka bot akan
pulang melalui jarak biasa . Jika kebalikannya, bot akan melalui jalur
teleportasi.

if jarak_base_biasa < jarak_base_teleportasi:
    self.goal_position ← base

else:
    if not(position_equals(sort_base_to_teleport[1].position,
current_position)):
        self.goal_position ← sort_base_to_teleport[1].position
    else:
        self.goal_position ← base

else:
    self.goal_position ← None

//Membuat list objek objek yang harus dihindari pada kasus tertentu contohnya
adalah teleportasi

```

```

dihindari ← []

//Ini digunakan untuk kembali ke base

if self.goal_position:
    penggabungan ← teleportasi
    for unsur in penggabungan:
        if not(position_equals(unsur.position, self.goal_position)):
            dihindari.append(unsur.position)

            delta_x, delta_y ← get_direction_Adv(current_position.x,
current_position.y, self.goal_position.x, self.goal_position.y, dihindari)

//Ini digunakan untuk bergerak ke diamond tujuan

else:
    penggabungan ← teleportasi

    for unsur in penggabungan:
        if not(position_equals(unsur.position, selected_goal.position)):
            dihindari.append(unsur.position)

            delta_x, delta_y ← get_direction_Adv(current_position.x,
current_position.y, selected_goal.position.x, selected_goal.position.y, dihindari)

filtered_arr_2 ← []
dihindari ← []

//Mengembalikan nilai pergerakan bot ke suatu tujuan atau objek
→ delta_x, delta_y

```

## **4.2 Penjelasan Struktur Data pada Program**

Dalam program ini, terdapat beberapa struktur data utama yang digunakan dalam pembuatan algoritma secara langsung. Struktur data tersebut adalah sebagai berikut:

### **1. Objek GameObject**

Objek GameObject ini adalah instance dari class GameObject yang berada pada board permainan dan dapat berinteraksi satu sama lain sesama objek GameObject. GameObject memiliki beberapa atribut seperti posisi di dalam papan, id, dan tipe objek. GameObject ini memiliki properties masing-masing sesuai dengan tipe objeknya. Objek yang ada adalah DiamondGameObject (diamond), TeleportGameObject (teleporter), BotGameObject (bot), dan DiamondButtonGameObject (red button). Properties dari diamond yang kami gunakan adalah posisi dan juga points dari diamond. Sedangkan untuk GameObject bertipe bot, teleporter, dan red button hanya membutuhkan atribut posisi object tersebut.

### **2. Objek Board**

Objek Board ini adalah instance dari class Board yang memiliki atribut id, panjang dan lebar papan permainan, list GameObject yang berada pada papan, delay minimum gerakan robot, dan list fitur yang ada pada papan. Adapun atribut yang kami manfaatkan secara langsung hanyalah config fitur pada list fitur yaitu inventory size sebagai salah satu pertimbangan dalam pemilihan langkah atau penetapan tujuan.

## **4.3 Analisis Desain Solusi Algoritma**

Solusi algoritma greedy yang digunakan pada permainan ini cukup optimal pada banyak kasus. Algoritma greedy dimulai dengan membuat *instance* dari kumpulan *class* untuk membentuk objek - objek utama yang digunakan dalam permainan seperti bot, diamond, tombol merah, dan lain-lain. Langkah selanjutnya adalah melakukan pengumpulan informasi berikut :

- a. Jarak bot utama terhadap semua diamond yang ada
- b. Jarak bot terdekat untuk setiap diamond tersebut

- c. *Density* setiap diamond dengan cara membagi poin diamond dengan jarak bot utama terhadap diamond.

Dua informasi di atas digunakan untuk menyaring diamond yang menjadi kandidat target. Diamond yang menjadi kandidat target adalah diamond yang jika bot utama bisa mencapai lokasi diamond tersebut terlebih dahulu dibandingkan bot musuh terdekat terhadap diamond tersebut. Hal tersebut dapat diketahui dengan mencari diamond yang jarak bot utama terhadap diamond tersebut dikurangi jarak bot terdekat terhadap diamond yang dimaksud bernilai negatif. Jika tidak ada satupun diamond yang memenuhi kriteria yang dijelaskan sebelumnya, sementara bot akan digerakkan menuju tombol merah dengan harapan diamond yang diproduksi ulang bisa menjadi kandidat solusi. Jika kandidat target diamond sudah ada, akan dikumpulkan informasi mengenai *density* terbesar bot musuh terhadap setiap diamond yang menjadi kandidat solusi. Berdasarkan informasi tersebut target utama dari bot utama bisa didapat dengan cara mencari diamond yang nilai *density* diamond tersebut terhadap bot utama lebih besar dibandingkan *density* bot musuh terbesar terhadap diamond yang dimaksud. Apabila ada dua atau lebih diamond yang memiliki kesamaan terkait nilai *density* terhadap bot utama, diamond yang terpilih adalah diamond dengan jarak terdekat.

Ada tiga kondisi bot harus kembali ke base. Pertama adalah ketika bot sudah membawa 4 poin di dalam *inventory* tapi diamond solusi yang harus diambil bernilai 2 poin. Kedua adalah ketika bot sudah membawa 5 poin yang artinya *inventory* sudah penuh. Ketiga adalah ketika langkah untuk kembali ke base sama dengan sisa waktu yang tersisa. Kondisi ketiga dilakukan untuk memastikan bot kembali ke base di akhir permainan dengan harapan bot juga membawa beberapa poin tambahan.

#### 4.4 Kasus Pengujian

Pada saat robot berjarak 2 kotak dari musuh, robot akan memiliki kemungkinan untuk mendekati musuh tersebut jika target posisi kami memang perlu melalui jalur tersebut. Jadi, pada saat robot mendekati musuh menjadi berjarak 1 kotak, robot berkemungkinan besar akan di-*tackle* oleh musuh jika

musuh tersebut menangani hal tersebut. Sehingga pada kasus tersebut, algoritma ini kurang optimal.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Ada banyak kasus atau kombinasi kejadian yang mungkin terjadi pada permainan Diamonds. Untuk mendapatkan kemenangan pada permainan tersebut, algoritma greedy dapat diimplementasikan untuk mendapatkan langkah yang terbaik untuk mengumpulkan poin selama permainan. Algoritma greedy yang dapat digunakan juga tidak hanya satu, tetapi ada beberapa kemungkinan tergantung dari situasi dan kondisi. Dengan melakukan pertimbangan terhadap efisiensi, efektivitas, dan risiko dari kumpulan alternatif algoritma greedy yang ada, kami memutuskan untuk memilih diamond berdasarkan *density* terbesar sebagai solusi terbaik. Hal tersebut juga didukung oleh beberapa percobaan yang kami lakukan. Algoritma greedy memilih diamond berdasarkan *density* lebih sering memenangkan pertandingan jika dibandingkan alternatif solusi yang lain.

#### **5.2. Saran**

Sebaiknya proses eksplorasi dan pemahaman yang mendalam terhadap permasalahan adalah hal yang perlu diprioritaskan sebelum pengambilan keputusan dan implementasi. Ada banyak *test case* atau kombinasi unika yang mungkin terjadi pada suatu persoalan khususnya pada permainan Diamonds. Mungkin saja kombinasi-kombinasi unik dan aneh tersebut akan menjadi parameter yang krusial untuk memutuskan algoritma greedy yang digunakan.

## **DAFTAR PUSTAKA**

1. Munir, Rinaldi. "Algoritma Greedy (Bagian 1)" (online).  
[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), diakses pada 24 Februari 2024).
2. Munir, Rinaldi. "Algoritma Greedy (Bagian 2)" (online).  
[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), diakses pada 24 Februari 2024).
3. Munir, Rinaldi. "Algoritma Greedy (Bagian 3)" (online).  
[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), diakses pada 24 Februari 2024).

## **LAMPIRAN**

### **Repository**

Link Repository dari Tugas Besar 01 IF2211 Strategi Algoritma kelompok 39 “Jadi Mesin” adalah sebagai berikut.

[https://github.com/PanjiSri/Tubes1\\_JadiMesin.git](https://github.com/PanjiSri/Tubes1_JadiMesin.git)

### **Youtube**

Link video Youtube dari Tugas Besar 01 IF2211 Strategi Algoritma kelompok 39 “Jadi Mesin” adalah sebagai berikut.

<https://youtu.be/VO88ukMsTak?si=JmWDwORMNHnueRrl>