

# **LAPORAN TUGAS BESAR STRATEGI ALGORITMA**

**Disusun oleh:  
Kelompok 12 - Ahajo**

**Dosen Pengampu : Winda Yulita, M.Cs.  
Asisten pembimbing : Meazza Ap**

Disusun oleh:  
M. Fadhil Hawari 123140147  
Abel Fortino 123140111  
Jonathan Nicholas Damero Sinaga 123140153

**TEKNIK INFORMATIKA  
INSTITUT TEKNOLOGI SUMATERA  
2025**

## KATA PENGANTAR

Puji syukur syukur kami panjatkan hadirat Allah SWT, karena atas rahmat dan karunia-Nya, sehingga kami dapat menyelesaikan Laporan Tugas Besar Strategi Algoritma ini. Makalah ini disusun sebagai salah satu tugas mata kuliah Algoritma, dengan fokus pada penerapan konsep Algoritma Greedy dengan keenam elemen-elemennya termasuk suatu jenis metode Algoritma Greedy yang dipakai. Kami menyampaikan rasa terima kasih yang sebesar besarnya kepada:

1. Ibu Dosen Pengampu mata kuliah Strategi Algoritma yang telah memberikan ilmu, bimbingan dan kesempatan bagi kami untuk menyelesaikan tugas ini.
2. Asisten Dosen yang memberikan arahan dan masukan selama Asistensi untuk kami dapat menyelesaikan tugas ini sesuai ketentuan yang diminta.
3. Rekan-rekan kelompok yang telah bekerja sama hingga laporan ini selesai.
4. Serta seluruh pihak lain yang turut membantu secara langsung maupun tidak langsung.

Semoga makalah ini dapat memberikan kontribusi positif dalam hal pemahaman konsep Algoritma Greedy. Akhir kata, kami menyampaikan permohonan maaf atas segala keterbatasan dalam makalah ini, Kami menyadari bahwasannya laporan ini masih jauh dari kata sempurna dan masih memiliki kekurangan dan kami menerima dengan terbuka segala kritik dan saran yang bersifat membangun

## DAFTAR ISI

<b>KATA PENGANTAR.....</b>	<b>1</b>
<b>BAB I.....</b>	<b>3</b>
1.1. Abstraksi.....	3
<b>BAB II.....</b>	<b>6</b>
2.1 Dasar Teori.....	6
2.2 Cara Kerja Program.....	7
<b>BAB III.....</b>	<b>9</b>
3.1 Proses Mapping Persoalan Diamonds ke Elemen-Element Algoritma Greedy.....	9
3.1.1 Himpunan Kandidat.....	9
3.1.2 Himpunan Solusi.....	9
3.1.3 Fungsi Solusi.....	9
3.1.4 Fungsi Seleksi.....	9
3.1.5 Fungsi Kelayakan.....	10
3.1.6 Fungsi Objektif.....	10
3.2 Eksplorasi alternatif solusi greedy yang mungkin dipilih dalam persoalan Diamonds.....	11
3.3 Analisis efisiensi dan efektifitas dari kumpulan alternatif solusi greedy yang dirumuskan.....	11
3.4 Strategi greedy yang dipilih.....	12
<b>BAB IV.....</b>	<b>13</b>
4.1 Implementasi algoritma greedy pada program bot yang digunakan.....	13
4.2 Penjelasan Struktur Data pada Program.....	15
4.3 Analisis Desain Solusi Algoritma.....	15
<b>BAB V.....</b>	<b>17</b>
<b>Daftar Pustaka.....</b>	<b>18</b>

# 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.

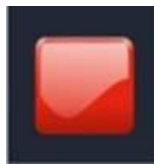
### 1) Diamonds



**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 Dasar Teori

Algoritma greedy merupakan pendekatan yang banyak digunakan dalam menyelesaikan masalah optimasi di bidang ilmu komputer. Pendekatan ini bekerja dengan membuat keputusan berdasarkan pilihan terbaik pada saat itu, atau secara lokal, dengan harapan bahwa rangkaian pilihan tersebut akan mengarah pada solusi yang optimal secara global. Dengan kata lain, algoritma greedy memilih opsi yang tampak paling menjanjikan pada setiap langkah tanpa mempertimbangkan kembali keputusan sebelumnya, sehingga menawarkan efisiensi dalam komputasi dibandingkan metode lain seperti dynamic programming (Cormen et al., 2009).

Prinsip utama algoritma greedy terletak pada dua karakteristik penting, yaitu *greedy-choice property* dan *optimal substructure*. *Greedy-choice property* mengacu pada sifat bahwa pilihan lokal terbaik pada setiap langkah dapat menjadi bagian dari solusi optimal global tanpa perlu mengevaluasi ulang keputusan sebelumnya. Misalnya, dalam suatu masalah, algoritma greedy akan memilih langkah yang memberikan keuntungan terbesar saat itu, dengan keyakinan bahwa pilihan ini akan berkontribusi pada solusi terbaik secara keseluruhan. Sementara itu, *optimal substructure* menunjukkan bahwa solusi optimal untuk masalah utama mengandung solusi optimal untuk sub masalahnya. Dengan kata lain, solusi global yang optimal dapat dibangun secara bertahap dari solusi optimal sub masalah yang lebih kecil (Cormen et al., 2009). Kedua sifat ini menjadi fondasi yang memungkinkan algoritma greedy bekerja secara efisien pada masalah tertentu.

Namun, penting untuk dicatat bahwa algoritma greedy tidak selalu menghasilkan solusi optimal untuk semua jenis masalah optimasi. Keberhasilannya bergantung pada apakah masalah yang dihadapi memenuhi *greedy-choice property* dan *optimal substructure*. Contoh masalah yang cocok untuk algoritma greedy adalah *activity-selection problem*, di mana tujuannya adalah memilih jumlah maksimum aktivitas yang saling kompatibel dari sekumpulan aktivitas dengan waktu mulai dan selesai tertentu. Dalam masalah ini, algoritma greedy memilih aktivitas dengan waktu selesai paling awal secara berulang, sehingga memaksimalkan jumlah aktivitas yang dapat dijadwalkan tanpa tumpang tindih. Pendekatan ini terbukti efisien dengan kompleksitas waktu  $O(n \log n)$  setelah pengaturan awal (Cormen et al., 2009). Contoh lain termasuk algoritma Kruskal dan Prim untuk menentukan pohon rentang minimum (*minimum spanning tree*) dalam graf berbobot, serta algoritma Huffman untuk kompresi data, yang membangun kode prefiks berdasarkan frekuensi karakter.

Keunggulan algoritma greedy terletak pada kesederhanaan dan efisiensi komputasinya. Dibandingkan dengan dynamic programming, yang mempertimbangkan semua kemungkinan solusi untuk submasalah, algoritma greedy hanya fokus pada pilihan lokal terbaik, sehingga sering kali memiliki kompleksitas waktu yang lebih rendah, seperti  $O(n \log n)$  untuk banyak kasus. Selain itu, implementasinya cenderung lebih sederhana, membuatnya menarik untuk masalah dengan skala besar. Namun, algoritma greedy juga memiliki keterbatasan. Pendekatan ini tidak selalu menghasilkan solusi optimal, terutama pada masalah yang tidak memenuhi *greedy-choice property*, seperti masalah *knapsack*.

0-1, di mana dynamic programming lebih sesuai. Oleh karena itu, analisis mendalam diperlukan untuk memastikan bahwa masalah yang dihadapi cocok dengan pendekatan greedy (Kleinberg & Tardos, 2005).

## 2.2 Cara Kerja Program

1) Bot mengelompokkan semua objek menjadi: diamonds, bots, teleport.

```
def get_obj_pos(self, board: Board):  
    self.diamonds = []  
    self.bots = []  
    self.teleport = []
```

2) Mencari teleportasi terdekat dan terjauh dari posisi bot saat ini.

```
def find_teleport_distance(self, x, y) -> tuple[GameObject, GameObject]
```

3) Cek inventory penuh dengan melihat apakah bot membawa diamond sebanyak kapasitas maksimum

```
if board_bot.properties.diamonds == board_bot.properties.inventory_size:
```

4) Cek jarak

Apakah musuh apakah dalam jarak < 2 unit

```
if self.check_enemy(board_bot, board)[1]:  
    goals_pos = self.check_enemy(board_bot, board)[0]
```

Jika Ya maka bergerak ke musuh

Jika Tidak Bandingkan antara jalur ke base langsung atau lewat teleportasi

Jika teleportasi lebih cepat:

```
goals_pos = nearest_teleport.position
```

Jika tidak

```
goals_pos = board_bot.properties.base
```

5) Hitung berapa diamond lagi yang bisa dibawa

```
Inventory_space = board_bot.properties.inventory_size -  
board_bot.properties.diamonds
```

6) Pilih diamond terdekat yang nilai poin-nya  $\leq$  sisa kapasitas

```
for d in self.diamonds:  
    // cek semua diamond  
  
    diamond_distance = ...  
    // hitung jarak total dengan bantuan teleport  
  
    if diamond_distance < min_dist:  
        // jika jalur teleport lebih pendek
```



```
point = ...  
if point <= inventory_space:  
    // jika inventory cukup menampung diamond  
    goals_pos = nearest_teleport.position  
    // arahkan bot ke teleport terdekat  
    min_dist = diamond_distance  
    // simpan jarak minimum baru
```

7) Keputusan akhir yaitu ke mana harus bergerak (kiri, kanan, atas, bawah)

```
if(board_bot.position.x > goals_pos.x):  
    delta_x = -1  
    delta_y = 0  
elif(board_bot.position.x < goals_pos.x):  
    delta_x = 1  
    delta_y = 0  
elif(board_bot.position.y > goals_pos.y):  
    delta_x = 0  
    delta_y = -1  
else:  
    delta_x = 0  
    delta_y = 1
```

## **BAB III**

### **APLIKASI STRATEGI GREEDY**

#### **3.1 Proses Mapping Persoalan Diamonds ke Elemen-Elemen Algoritma Greedy**

Algoritma greedy memecahkan masalah optimasi dengan membuat pilihan lokal terbaik pada setiap langkah, dengan harapan menghasilkan solusi global yang optimal (Cormen et al., 2009). Pendekatan greedy dalam persoalan Diamonds harus dilakukan pemetaan elemen elemen masalah yaitu:

##### **3.1.1 Himpunan Kandidat**

Himpunan kandidat merupakan semua kemungkinan langkah atau tujuan yang dapat dipilih oleh bot di setiap iterasi. Dalam persoalan game diamonds, himpunan kandidatnya adalah:

1. Posisi berlian yang dapat diambil
2. Posisi teleport
3. Posisi base atau tempat asal
4. Posisi bot musuh

Masing-masing kandidat memiliki bentuk  $(x,y)$  yang merupakan koordinat mereka di bidang kartesian

##### **3.1.2 Himpunan Solusi**

Himpunan solusi adalah urutan langkah-langkah yang diambil oleh bot untuk mencapai tujuan, yaitu mendapatkan skor sebanyak mungkin dan kembali ke base, sehingga dalam konteks game diamonds himpunan solusi merupakan daftar posisi yang dikunjungi oleh bot dalam bentuk koordinat. Solusi akhir dari himpunan solusi merupakan jalur lengkap atau daftar koordinat yang dilalui bot untuk memaksimalkan skor. Dalam hal ini daftar himpunan solusinya berupa:

1. Posisi diamond yang terdekat dan di saat cukup untuk dimasukkan ke inventory
2. Posisi musuh (jika terlalu dekat)
3. Posisi teleport yang mempercepat atau jarak terdekat menuju base atau diamond
4. Posisi base jika inventory penuh

Himpunan solusi dihasilkan dengan mempertimbangkan kombinasi

1. Jarak ke Object
2. Kapasitas inventory (masih mencukupi atau sudah penuh)
3. Jarak ke musuh

##### **3.1.3 Fungsi Solusi**

Fungsi solusi merupakan fungsi yang menentukan kapan suatu masalah dapat dianggap selesai, sehingga dalam permainan diamonds kita memiliki fungsi solusi sebagai berikut:

1. Inventory bot sudah penuh dan bot sudah kembali ke base
2. Berlian yang tersedia di papan sudah habis dan bot sudah kembali ke base
3. Waktu permainan habis

##### **3.1.4 Fungsi Seleksi**

Fungsi seleksi menjelaskan bagaimana algoritma memilih kandidat terbaik pada setiap langkah. Dalam bot yang kami buat fungsi seleksinya adalah sebagai berikut:

1. Bot mencari berlian terdekat berdasarkan jarak euclidean yang poinnya tidak melebihi kapasitas inventory.
2. Bot mencari teleportasi terdekat berdasarkan jarak euclidean dengan asumsi bahwa hal tersebut akan mempersingkat jarak terhadap diamond
3. Jika inventory penuh bot memilih
  - Langsung menuju base, atau
  - Menggunakan kombinasi teleportasi yaitu sebagai pintu masuk dan teleport terjauh sebagai pintu keluar menuju base, lalu membandingkan total jarak dengan rute langsung . Bot memilih mana jalur yang paling pendek/singkat
4. Jika ada musuh berada di jarak yang ditentukan, maka bot mengutamakan bergerak menuju posisi musuh sebagai bentuk prioritas untuk menghindari atau menghadapi musuh
5. Pemilihan jalur ke berlian atau teleportasi di prioritaskan melalui
  - Jarak ke berlian melalui teleport lebih pendek daripada jarak langsung
  - Poin berlian masih muat di inventory, maka bot akan mengakses berlian melalui jalur teleportasi

### **3.1.5 Fungsi Kelayakan**

Fungsi kelayakan menentukan apakah suatu kandidat yang dipilih dapat ditambahkan ke himpunan solusi parsial yang sedang dibangun. Fungsi ini memeriksa apakah sebuah langkah atau pilihan memenuhi syarat-syarat tertentu pada kondisi saat itu. Suatu langkah dikatakan layak jika memenuhi beberapa syarat berikut:

1. Teleportasi yang dipilih memiliki koordinat tujuan yang valid dan bukan terblokir atau sudah ditempati.
2. Tidak berisiko terlalu dekat dengan musuh (misalnya dalam radius bahaya tertentu).
3. Langkah tersebut tidak mengakibatkan bot terjebak atau jauh dari base ketika waktu hampir habis.
4. Posisi tujuan masih ada di papan permainan dan belum diambil/diklaim oleh bot lain.

### **3.1.6 Fungsi Objektif**

Fungsi objektif merupakan tujuan akhir dari algoritma yang ingin dicapai melalui serangkaian langkah-langkah yang diambil. Fungsi objektif dari algoritma greedy yang digunakan oleh bot adalah:

1. Meminimalkan jarak tempuh ke diamond, teleport, atau base, sehingga waktu tidak terbuang dan bot bisa melakukan lebih banyak aksi dalam satu pertandingan.
2. Menggunakan teleportasi secara strategis bila memungkinkan mempercepat rute ke diamond atau ke base ketika inventory penuh.
3. Mengosongkan inventory di base secara berkala agar tetap bisa mengangkut diamond baru tanpa membuang langkah sia-sia.
4. Kembali langsung ke base bila jarak ke teleportasi lebih jauh dibandingkan ke base dengan tepat waktu jika inventory penuh

### 3.2 Eksplorasi alternatif solusi greedy yang mungkin dipilih dalam persoalan Diamonds

Dalam pengembangan strategi greedy untuk bot pengumpul diamond, terdapat beberapa pendekatan yang dapat dipertimbangkan untuk meningkatkan performa bot dalam pengumpulan diamonds seperti:

1. Greedy by Diamond Value Density
  - Memprioritaskan diamond dengan nilai tertinggi per unit jarak
  - Lebih efisien dalam meningkatkan skor per langkah
2. Greedy with Time Constraint
  - Memastikan bot punya cukup waktu untuk kembali ke base
  - Mencegah bot terjebak terlalu jauh dari base
3. Greedy with Aggressive Collection
  - Mencari cluster diamond terdekat
  - Memaksimalkan pengumpulan per trip
4. Hybrid Greedy Approach
  - Menyeimbangkan berbagai faktor penting
  - Dapat disesuaikan bobot parameternya
5. Greedy with Risk Avoidance
  - Meminimalisir konflik dengan bot lain
  - Lebih konservatif dalam menghindari risiko

### 3.3 Analisis efisiensi dan efektifitas dari kumpulan alternatif solusi greedy yang dirumuskan

Dalam menganalisis berbagai pendekatan greedy untuk bot pengumpul diamond, kita dapat melihat trade-off yang menarik antara efisiensi komputasi dan efektivitas strategi seperti:

1. Greedy by Diamond Value Density  
Efisiensi:
  - Kompleksitas tetap  $O(n)$  untuk mengevaluasi semua diamond
  - Perhitungan value/distance sederhana namun berdampak besar
  - Optimal untuk papan dengan distribusi nilai diamond bervariasiEfektivitas:
  - Maksimisasi poin per unit waktu
  - Cocok untuk early game saat inventory masih kosong
2. Greedy with Time Constraint  
Efisiensi:
  - Memerlukan perhitungan jarak ganda (ke diamond + ke base)
  - Kompleksitas  $O(2n) \approx$  masih linear
  - Overhead minimal untuk pengecekan waktuEfektivitas:
  - Mencegah game over karena waktu habis
  - Sangat efektif di late game
3. Cluster-based Aggressive Collection  
Efisiensi:

- Kompleksitas  $O(n^2)$  untuk deteksi cluster
- Resource-intensive untuk papan besar
- Cocok untuk papan dengan diamond terkonsentrasi

Efektivitas:

- Optimalisasi movement dengan multi-collection
- Efek domino positif pada skor akhir

#### 4. Hybrid Greedy Approach

Efisiensi:

- Perhitungan multifaktor ( $0.4 \cdot \text{dist} + 0.4 \cdot \text{value} + 0.2 \cdot \text{time}$ )
- Kompleksitas tetap  $O(n)$
- Fleksibel melalui penyesuaian bobot

Efektivitas:

- Balance antara berbagai strategi
- Adaptif terhadap fase permainan
- Membutuhkan tuning bobot yang presisi

#### 5. Risk-Averse Greedy

Efisiensi:

- $O(n \cdot m)$  untuk  $n$  diamond dan  $m$  bot
- Mahal secara komputasi
- Cocok untuk multiplayer competitive

Efektivitas:

- Minimasi konflik dengan bot lain
- Konsistensi pengumpulan
- Trade-off: mungkin melewatkan opportunity

### 3.4 Strategi greedy yang dipilih

Strategi utama yang diterapkan dalam bot ini mengadopsi pendekatan greedy berbasis jarak terdekat dengan pertimbangan keamanan dari bot lawan dan teleportasi. Bot secara konsisten memilih diamond terdekat yang sesuai dengan kapasitas inventory saat ini, dengan mekanisme fallback ke penghindaran musuh ketika terdeteksi bahaya. Implementasi ini menggabungkan tiga lapisan strategi:

1. prioritas keamanan melalui pengecekan musuh dalam radius 2 unit
2. optimasi rute menggunakan portal teleport ketika menguntungkan
3. default ke algoritma greedy konvensional pencarian diamond terdekat

Dari segi efektivitas, strategi ini mengutamakan dalam beberapa aspek seperti:

1. Manajemen risiko dengan melakukan penghindaran musuh yang dimulai ketika inventory penuh
2. Optimalisasi gerakan melalui analisis biaya-manfaat penggunaan teleport yang dihitung berdasarkan jarak antara Bot dengan teleportasi dan jarak teleportasi dengan GameObject lain seperti Diamond maupun Base
3. Keputusan yang berbeda saat inventory kosong dan saat penuh

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi algoritma greedy pada program bot yang digunakan

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

```
CLASS HCLBot EXTENDS BaseLogic

METHOD __init__():
    // Konstruktor kosong

METHOD get_obj_pos(board):
    // Ambil semua objek di papan permainan
    INISIALISASI diamonds SEBAGAI list kosong
    INISIALISASI bots SEBAGAI list kosong
    INISIALISASI teleport SEBAGAI list kosong

    UNTUK setiap obj DALAM board.game_objects:
        JIKA obj adalah Diamond ATAU DiamondButton:
            TAMBAHKAN obj KE diamonds
        JIKA obj adalah Bot (termasuk bot musuh):
            TAMBAHKAN obj KE bots
        JIKA obj adalah Teleport:
            TAMBAHKAN obj KE teleport

METHOD calculate_distance(x1, y1, x2, y2) -> float:
    // Hitung jarak Euclidean antara dua titik
    RETURN sqrt((x2 - x1)^2 + (y2 - y1)^2)

METHOD check_enemy(board_bot, board) -> (Position, bool):
    // Periksa apakah ada bot musuh di dekat bot kita (radius < 2)
    UNTUK setiap bot DALAM bots:
        JIKA bot.id BUKAN bot kita DAN jaraknya < 2:
            RETURN (bot.position, True) // musuh dekat
    RETURN (bot.position, False) // tidak ada musuh dekat

METHOD find_teleport_distance(x, y) -> (GameObject, GameObject):
    // Temukan teleport terdekat dan terjauh dari posisi (x, y)
    SET min_dist = ∞
    SET max_dist = -∞
    SET nearest_teleport = None
    SET farthest_teleport = None

    UNTUK setiap teleport DALAM daftar teleport:
        HITUNG dist = jarak ke teleport
        JIKA dist < min_dist:
```

```

    nearest_teleport = teleport
    min_dist = dist
    JIKA dist > max_dist:
        farthest_teleport = teleport
        max_dist = dist

    RETURN (nearest_teleport, farthest_teleport)

METHOD next_move(board_bot, board) -> (delta_x, delta_y):
    PANGGIL get_obj_pos(board)
    SET min_dist = ∞
    SET goals_pos = posisi diamond pertama (default)
    PANGGIL find_teleport_distance(...) UNTUK dapatkan teleport terdekat & terjauh

    // Jika inventory penuh
    JIKA jumlah diamond yang dibawa == kapasitas maksimal:
        JIKA ada musuh dekat:
            goals_pos = posisi musuh
        LAIN:
            HITUNG base_distance (tanpa teleport)
            HITUNG base_distance_teleport (pakai teleport)
            JIKA jarak teleport lebih cepat:
                goals_pos = posisi teleport terdekat
            LAIN:
                goals_pos = base (markas)

    // Jika inventory belum penuh
    LAIN:
        inventory_space = sisa ruang inventory

    // Cari diamond terdekat (tanpa teleport)
    UNTUK setiap diamond:
        JIKA ada musuh dekat:
            goals_pos = posisi musuh
        ELSE JIKA jarak ke diamond < min_dist DAN point diamond muat di inventory:
            goals_pos = posisi diamond
            min_dist = jarak

    // Cari diamond terdekat (pakai teleport)
    UNTUK setiap diamond:
        HITUNG diamond_distance = jarak dari bot ke teleport terdekat + dari teleport terjauh ke
diamond
        JIKA diamond_distance < min_dist DAN point diamond muat di inventory:
            goals_pos = posisi teleport terdekat
            min_dist = diamond_distance

    CETAK min_dist dan diamond_distance (debug)

    // Gerak menuju goals_pos (delta_x dan delta_y 1 langkah)
    JIKA posisi bot.x > goals_pos.x:

```

```

    delta_x = -1
    delta_y = 0
ELSE JIKA posisi bot.x < goals_pos.x:
    delta_x = 1
    delta_y = 0
ELSE JIKA posisi bot.y > goals_pos.y:
    delta_x = 0
    delta_y = -1
ELSE:
    delta_x = 0
    delta_y = 1

RETURN (delta_x, delta_y)

```

## 4.2 Penjelasan Struktur Data pada Program

Struktur data yang digunakan dalam program ini menggunakan dua object utama dalam pembuatan algoritma secara langsung. Struktur data tersebut adalah sebagai berikut:

### 1. Objek GameObject

GameObject merupakan object yang mempresentasikan sebagai entitas berbagai elemen dalam game. Terdapat berbagai entitas elemen seperti DiamondGameObject(diamond yang bisa diambil), BotGameObject(bot pemain), dan TeleportGameObject (teleportasi). Setiap GameObject memiliki elemen atau properti seperti type (jenis object), id (identitas unik), position (koordinat x dan y), dan juga properties yang berisi atribut tambahan seperti diamonds (jumlah diamond yang dibawa oleh bot), inventory\_size (kapasitas inventory) dan points (nilai dari diamond)

### 2. Objek Board

Board berfungsi sebagai peta permainan yang menyimpan semua Game Object dalam bentuk GameObjectlist game\_objects. Program kemudian mengelompokkan objek-objek ini ke dalam kategori seperti diamonds, bots, dan teleport untuk memudahkan pemrosesan. Logika permainan diimplementasikan dengan menghitung jarak antar-objek, memeriksa keberadaan musuh, serta menentukan gerakan optimal berdasarkan posisi diamond terdekat, penggunaan teleport, atau kondisi inventory. Jika inventory penuh, bot akan kembali ke base, sedangkan jika ada musuh di dekatnya, bot akan menghindar atau menyerang. Dengan struktur data ini, bot dapat mengambil keputusan secara dinamis berdasarkan keadaan permainan.

## 4.3 Analisis Desain Solusi Algoritma

Dalam program ini, kita menggunakan dua struktur data utama sebagai fondasi algoritmanya. Pertama ada GameObject yang berfungsi merepresentasikan berbagai elemen dalam game. Setiap GameObject ini punya karakteristik berbeda-beda - ada yang berupa diamond bisa dikumpulkan (DiamondGameObject), ada yang merupakan bot pemain (BotGameObject), sampai portal teleportasi (TeleportGameObject). Masing-masing object ini dilengkapi properti dasar seperti:



- Type untuk mengetahui jenis objectnya
- ID sebagai identitas unik
- Posisi dalam koordinat x dan y
- Berbagai properti tambahan seperti jumlah diamond yang dibawa, kapasitas inventory, sampai nilai point dari diamond

Kedua ada Board yang bertindak sebagai peta permainan. Board inilah yang menyimpan semua GameObject dalam sebuah list. Untuk mempermudah pengolahan data, program kemudian mengelompokkan object-object ini menjadi beberapa kategori seperti diamonds, bots, dan teleport.

Dari struktur data ini, kemudian dibangun logika permainan yang cukup kompleks. Sistem akan menghitung jarak antar objek, memantau keberadaan musuh di sekitar, lalu menentukan langkah terbaik berdasarkan beberapa faktor:

1. Posisi diamond terdekat yang bisa diambil
2. Ketersediaan dan posisi teleport
3. Kapasitas inventory saat ini

Ketika inventory sudah penuh, bot otomatis akan pulang ke base. Tapi kalau ada musuh mendekat dalam jarak berbahaya, bot akan lebih memprioritaskan untuk menghindar. Fleksibilitas inilah yang membuat bot bisa beradaptasi dengan dinamika permainan.

Yang menarik, sistem perhitungan jaraknya menggunakan pendekatan praktis. Untuk menentukan rute tercepat, bot akan membandingkan jarak normal dengan alternatif menggunakan teleport. Keputusan diambil berdasarkan mana yang lebih efisien waktu.

Dari segi implementasi, kode ini menunjukkan pendekatan terstruktur namun tetap fleksibel. Pengelompokan object mempermudah proses pencarian, sementara logika kondisi yang berlapis memungkinkan pengambilan keputusan yang kontekstual. Meski begitu, tetap ada ruang untuk pengembangan lebih lanjut terutama dalam optimasi pathfinding.

## **BAB V**

### **KESIMPULAN DAN SARAN**

Setelah menyelesaikan implementasi strategi algoritma greedy dalam permainan Diamonds, kami menyimpulkan bahwa strategi greedy ini cukup efektif untuk digunakan dalam situasi yang membutuhkan pengambilan keputusan cepat. Bot yang kami kembangkan mampu bertindak secara cerdas dengan memprioritaskan pengambilan diamond terdekat, memperhatikan kapasitas inventory, serta menghindari musuh yang berada terlalu dekat. Dengan menambahkan logika penggunaan teleportasi dan strategi berbeda saat inventory penuh atau kosong, bot menjadi lebih adaptif terhadap kondisi yang berubah-ubah di dalam permainan. Struktur data yang digunakan pun mendukung pengambilan keputusan yang efisien, sehingga bot dapat bergerak secara optimal di papan permainan tanpa membuang banyak waktu. Meskipun begitu, kami menyadari bahwa pendekatan greedy memiliki batasan, terutama karena keputusan yang diambil selalu bersifat lokal dan tidak mempertimbangkan dampak jangka panjang secara menyeluruh.

### **Daftar Pustaka**

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). MIT Press.
2. Kleinberg, J., & Tardos, É. (2005). *Algorithm design*. Pearson.