

# **IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND**

## **Tugas Besar**

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RB  
di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera



**Oleh: Kelompok 2 (Bakso Goreng)**

Dela Puspita Sari	123140080
I Kadek Maruta Pracheta	123140085
Ibrahim Budi Satria	123140097

Dosen Pengampu: Imam Ekowicaksono, S.Si., M.Si.

**PROGRAM STUDI TEKNIK INFORMATIKA  
FAKULTAS TEKNOLOGI INDUSTRI  
INSTITUT TEKNOLOGI SUMATERA  
2025**

## DAFTAR ISI

<b>BAB I DESKRIPSI TUGAS .....</b>	<b>3</b>
<b>BAB II LANDASAN TEORI .....</b>	<b>6</b>
2.1 Dasar Teori.....	7
2.2 Cara Kerja Program .....	7
2.2.1. Cara Implementasi Program .....	7
2.2.2. Menjalankan Bot Program .....	7
<b>BAB III APLIKASI STRATEGI GREEDY.....</b>	<b>10</b>
3.1 Proses Mapping.....	10
3.2 Eksplorasi Alternatif Solusi Greedy .....	10
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy.....	10
3.4 Strategi Greedy yang Dipilih .....	11
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN .....</b>	<b>12</b>
4.1 Implementasi Algoritma Greedy.....	12
4.1.1. Pseudocode .....	12
4.1.2. Penjelasan Alur Program .....	17
4.2 Struktur Data yang Digunakan.....	18
4.3 Pengujian Program.....	18
4.3.1. Skenario Pengujian .....	18
4.3.2. Hasil Pengujian dan Analisis .....	18
<b>BAB V KESIMPULAN DAN SARAN.....</b>	<b>19</b>
5.1 Kesimpulan .....	19
5.2 Saran .....	19
<b>LAMPIRAN.....</b>	<b>20</b>
<b>DAFTAR PUSTAKA.....</b>	<b>21</b>

# BAB I

## DESKRIPSI TUGAS

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



**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/melangkahnya. 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.

##### 1) 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 adalah metode yang sering digunakan dalam menyelesaikan masalah optimasi di bidang ilmu komputer. Pendekatan ini bekerja dengan memilih opsi terbaik di setiap langkah tanpa mempertimbangkan keputusan sebelumnya, dengan harapan keputusan-keputusan tersebut menghasilkan solusi optimal secara global. Algoritma ini lebih efisien dibanding metode seperti dynamic programming karena hanya fokus pada pilihan lokal terbaik (Cormen et al., 2009).

Agar efektif, algoritma greedy bergantung pada dua prinsip utama, yaitu greedy-choice property dan optimal substructure. Greedy-choice property berarti keputusan terbaik di tiap langkah bisa menjadi bagian dari solusi global tanpa perlu mengevaluasi ulang keputusan sebelumnya. Sementara optimal substructure menunjukkan bahwa solusi optimal suatu masalah terdiri dari solusi optimal submasalahnya (Cormen et al., 2009).

Meski begitu, algoritma greedy tidak selalu cocok untuk semua jenis masalah. Keberhasilannya tergantung terpenuhinya kedua prinsip tadi. Contohnya, activity-selection problem dapat diselesaikan dengan greedy karena cukup memilih aktivitas yang selesai paling awal secara berulang untuk memaksimalkan jumlah aktivitas tanpa bentrok, dengan waktu komputasi  $O(n \log n)$ . Algoritma ini juga digunakan pada algoritma Kruskal, Prim, dan Huffman coding.

Keunggulan algoritma greedy ada pada kesederhanaan konsep dan efisiensi komputasinya. Dibanding dynamic programming yang mempertimbangkan semua kemungkinan solusi, greedy hanya memilih opsi terbaik saat itu. Meski cepat, algoritma ini memiliki kelemahan karena tidak selalu menjamin solusi optimal, seperti pada 0-1 knapsack problem, yang lebih sesuai diselesaikan dengan dynamic programming (Kleinberg & Tardos, 2005). Oleh karena itu, perlu analisis masalah sebelum menentukan metode yang tepat.

## **2.2 Cara Kerja Program**

Secara umum, cara kerja program ini adalah mengendalikan sebuah bot yang bertugas untuk mengumpulkan diamond sebanyak mungkin di dalam sebuah area permainan. Bot akan bergerak dari satu titik ke titik lainnya dengan mempertimbangkan jarak terdekat ke diamond berikutnya. Untuk menentukan langkah yang diambil, program menggunakan algoritma greedy yang memprioritaskan pengambilan diamond dengan jarak terdekat dari posisi bot saat ini. Selain itu, bot juga dibekali fitur untuk memutuskan apakah perlu menggunakan fasilitas teleport yang tersedia untuk mempercepat pergerakan menuju lokasi diamond atau tetap bergerak manual. Program berjalan secara otomatis dengan menentukan keputusan terbaik di setiap langkah tanpa meninjau ulang keputusan sebelumnya, sesuai prinsip greedy.

### **2.2.1 Cara Implementasi Program**

Cara kerja program ini adalah dengan menerapkan algoritma greedy yang mempertimbangkan dua faktor utama, yaitu jarak ke diamond terdekat dan nilai yang diperoleh dari diamond tersebut. Pada setiap langkah, bot akan mencari diamond dengan jarak paling dekat dari posisinya saat ini. Jika ditemukan beberapa diamond dengan jarak yang sama, maka bot akan memilih diamond yang memberikan nilai paling tinggi. Selain itu, program juga dilengkapi fitur untuk memutuskan apakah perlu menggunakan teleport atau tidak. Bot akan mempertimbangkan opsi teleport jika jarak menuju diamond berikutnya lebih jauh daripada jarak ke titik teleport terdekat ditambah jarak dari teleport ke lokasi diamond. Dengan strategi ini, bot diharapkan dapat mengumpulkan diamond dalam jumlah maksimal dalam waktu yang lebih efisien. Proses implementasi dilakukan dengan mendefinisikan daftar posisi diamond, posisi bot, dan posisi teleport, lalu program akan secara iteratif memilih opsi terbaik di setiap langkah hingga semua diamond terkumpul atau waktu habis.

### **2.2.2 Menjalankan Bot Program**

- Menentukan posisi tujuan kemudian bot memutuskan arah gerak bot berikutnya. Jika diamond sudah 5 maka bot akan langsung cari jalan tercepat menuju base, jika waktu hampir habis dengan diamond minimal 2 bot akan langsung menuju base, jika dekat dengan base dan diamond minimal 3 maka bot akan menyimpan ke base.

```

1 def next_move(self, board_bot: GameObject, board: Board):
2     props = board_bot.properties
3     self.board = board
4     self.board_bot = board_bot
5     self.diamonds = board.diamonds
6     self.teleporter = [d for d in board.game_objects if d.type == "TeleportGameObject"]
7     self.redButton = [d for d in board.game_objects if d.type == "DiamondButtonGameObject"]
8
9     if board_bot.position == props.base:
10        self.static_goals.clear()
11        self.static_goal_teleport = None
12        self.static_temp_goals = None
13        self.static_direct_to_base_via_teleporter = False
14
15    if self.static_goal_teleport and board_bot.position == self.find_other_teleport(self.static_goal_teleport):
16        self.static_goals.remove(self.static_goal_teleport.position)
17        self.static_goal_teleport = None
18    if not self.static_goal_teleport and board_bot.position in self.static_goals:
19        self.static_goals.remove(board_bot.position)
20    if board_bot.position == self.static_temp_goals:
21        self.static_temp_goals = None
22
23    if props.diamonds == 5 or (props.milliseconds_left < 5000 and props.diamonds > 1):
24        self.goal_position = self.find_best_way_to_base()
25        if not self.static_direct_to_base_via_teleporter:
26            self.static_goals.clear()
27            self.static_goal_teleport = None
28        else:
29            if not self.static_goals:
30                self.find_nearest_diamond()
31                self.goal_position = self.static_goals[0]
32
33    if self.calculate_near_base() and props.diamonds > 2:
34        self.goal_position = self.find_best_way_to_base()
35        if not self.static_direct_to_base_via_teleporter:
36            self.static_goals.clear()
37            self.static_goal_teleport = None
38
39    if self.static_temp_goals:
40        self.goal_position = self.static_temp_goals
41
42    if self.goal_position:
43        if not self.static_temp_goals:
44            self.check_obstacle('teleporter')
45        if props.diamonds == 4:
46            self.check_obstacle('redDiamond')
47        dx, dy = get_direction(board_bot.position.x, board_bot.position.y,
48                               self.goal_position.x, self.goal_position.y)
49    else:
50        dx, dy = self.d

```

- Menentukan apakah lebih cepat ke base secara langsung atau lewat teleport. Jika teleport lebih efisien, tujuan sementara akan teleport.

```

1 def find_best_way_to_base(self):
2     cur = self.board_bot.position
3     base = self.board_bot.properties.base
4     base_pos = Position(base.y, base.x)
5     dist_direct = abs(base.x - cur.x) + abs(base.y - cur.y)
6
7     tp_in, tp_out, tp_obj = self.find_nearest_teleport()
8     if not tp_in or not tp_out:
9         return base_pos
10
11    dist_tp = (abs(tp_in.x - cur.x) + abs(tp_in.y - cur.y) +
12              abs(tp_out.x - base.x) + abs(tp_out.y - base.y))
13
14    if dist_direct <= dist_tp:
15        return base_pos
16
17    self.static_direct_to_base_via_teleporter = True
18    self.static_goal_teleport = tp_obj
19    self.static_goals = [tp_in, base_pos]
20    return tp_in

```



- Mencari diamond terdekat dengan mempertimbangkan 3 kondisi yaitu jalur langsung, melalui teleport jika lebih menguntungkan, dan red button jika ada.

```
1 def find_nearest_diamond(self):
2     direct_dist, direct_pos = self.find_nearest_diamond_direct()
3     tp_dist, tp_path, tp_obj = self.find_nearest_diamond_teleport()
4     red_dist, red_pos = self.find_nearest_red_button()
5
6     if direct_dist <= tp_dist and direct_dist <= red_dist:
7         self.static_goals = [direct_pos]
8         self.distance = direct_dist
9     elif tp_dist <= red_dist:
10        self.static_goals = tp_path
11        self.static_goal_teleport = tp_obj
12        self.distance = tp_dist
13    else:
14        self.static_goals = [red_pos]
15        self.distance = red_dist
```

## **BAB III**

### **APLIKASI STRATEGI *GREEDY***

#### **3.1 Proses *Mapping***

Proses mapping merupakan tahap awal yang penting dalam pengembangan program bot, di mana seluruh objek yang terdapat di area permainan dipetakan ke dalam sistem. Beberapa objek penting yang perlu dikenali oleh bot antara lain diamond, teleport, red button, dan base. Diamond berfungsi sebagai target utama yang harus dikumpulkan sebanyak mungkin. Teleport berperan sebagai alat bantu untuk berpindah lokasi dengan cepat, sementara red button merupakan objek yang dapat memengaruhi kondisi area permainan, misalnya generate ulang posisi diamond. Base adalah titik awal sekaligus titik akhir bagi bot untuk memulai dan menyelesaikan misi pengumpulan diamond. Semua posisi objek ini disimpan dalam bentuk koordinat pada peta virtual agar bot dapat mengenali letak masing-masing objek. Dengan mapping ini, bot dapat menentukan rute pergerakan berdasarkan posisi aktual objek yang tersedia di area permainan.

#### **3.2 Eksplorasi Alternatif Solusi Greedy**

Alternatif solusi greedy yang kami gunakan pada bot ini adalah :

- Greedy berdasarkan rasio poin/jarak
- Greedy berdasarkan jarak
- Greedy dengan teleport jika menguntungkan
- Greedy dengan prioritas red button jika tidak ada diamond terdekat

#### **3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy**

Berdasarkan analisis alternatif strategi, greedy dengan rasio point per jarak terbukti lebih efektif dibandingkan hanya berdasarkan jarak. Strategi ini mempertimbangkan keuntungan relatif di setiap langkah, sehingga meskipun jaraknya sedikit lebih jauh, nilai yang diperoleh lebih besar dan total point yang terkumpul lebih optimal.

### **3.4 Strategi Greedy yang Dipilih**

Dari berbagai alternatif strategi yang telah dianalisis sebelumnya, strategi yang akhirnya dipilih dalam program ini adalah greedy berdasarkan rasio nilai per jarak. Artinya, bot tidak hanya mempertimbangkan jarak terdekat semata, tetapi juga memperhitungkan nilai diamond yang akan diambil dibandingkan jarak yang harus ditempuh untuk mendapatkannya. Dengan cara ini, bot akan memilih diamond yang memberikan rasio nilai tertinggi terhadap jarak tempuh di setiap langkahnya. Strategi ini dipilih karena dinilai paling efektif dalam memaksimalkan total nilai diamond yang diperoleh dalam waktu terbatas, sekaligus tetap mempertahankan efisiensi pergerakan bot di area permainan.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Algoritma Greedy

##### 4.1.1 Pseudocode

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

class baksogorengg(BaseLogic):
    static_goals: list[Position] = []
    static_goal_teleport: Optional[GameObject] = None
    static_temp_goals: Optional[Position] = None
    static_direct_to_base_via_teleporter: bool = False

    def _init_(self) -> None:
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] = None
        self.current_direction = 0
        self.distance = 0

    def next_move(self, board_bot: GameObject, board: Board):
        props = board_bot.properties
        self.board = board
        self.board_bot = board_bot
        self.diamonds = board.diamonds
        self.teleporter = [d for d in board.game_objects if d.type ==
"TeleportGameObject"]
        self.redButton = [d for d in board.game_objects if d.type ==
"DiamondButtonGameObject"]

        if board_bot.position == props.base:
            self.static_goals.clear()
            self.static_goal_teleport = None
            self.static_temp_goals = None
            self.static_direct_to_base_via_teleporter = False
```

```

        if self.static_goal_teleport and board_bot.position ==
self.find_other_teleport(self.static_goal_teleport):
            self.static_goals.remove(self.static_goal_teleport.position)
            self.static_goal_teleport = None
        if not self.static_goal_teleport and board_bot.position in
self.static_goals:
            self.static_goals.remove(board_bot.position)
        if board_bot.position == self.static_temp_goals:
            self.static_temp_goals = None

        if props.diamonds == 5 or (props.milliseconds_left < 5000 and
props.diamonds > 1):
            self.goal_position = self.find_best_way_to_base()
            if not self.static_direct_to_base_via_teleporter:
                self.static_goals.clear()
                self.static_goal_teleport = None
        else:
            if not self.static_goals:
                self.find_nearest_diamond()
            self.goal_position = self.static_goals[0]

        if self.calculate_near_base() and props.diamonds > 2:
            self.goal_position = self.find_best_way_to_base()
            if not self.static_direct_to_base_via_teleporter:
                self.static_goals.clear()
                self.static_goal_teleport = None

        if self.static_temp_goals:
            self.goal_position = self.static_temp_goals

        if self.goal_position:
            if not self.static_temp_goals:
                self.check_obstacle('teleporter')
            if props.diamonds == 4:
                self.check_obstacle('redDiamond')
            dx, dy = get_direction(board_bot.position.x, board_bot.position.y,
                                self.goal_position.x, self.goal_position.y)
        else:
            dx, dy = self.directions[self.current_direction]
            self.current_direction = (self.current_direction + 1) %
len(self.directions)

```

```

        if dx == 0 and dy == 0:
            self.static_goals.clear()
            self.static_goal_teleport = None
            self.static_temp_goals = None
            self.goal_position = None
            dx, dy = self.next_move(board_bot, board)

    return dx, dy

def find_best_way_to_base(self):
    cur = self.board_bot.position
    base = self.board_bot.properties.base
    base_pos = Position(base.y, base.x)
    dist_direct = abs(base.x - cur.x) + abs(base.y - cur.y)

    tp_in, tp_out, tp_obj = self.find_nearest_teleport()
    if not tp_in or not tp_out:
        return base_pos

    dist_tp = (abs(tp_in.x - cur.x) + abs(tp_in.y - cur.y) +
               abs(tp_out.x - base.x) + abs(tp_out.y - base.y))

    if dist_direct <= dist_tp:
        return base_pos

    self.static_direct_to_base_via_teleporter = True
    self.static_goal_teleport = tp_obj
    self.static_goals = [tp_in, base_pos]
    return tp_in

def calculate_near_base(self):
    cur = self.board_bot.position
    base = self.board_bot.properties.base
    dist_direct = abs(base.x - cur.x) + abs(base.y - cur.y)
    dist_tp = self.find_base_distance_teleporter()
    return min(dist_direct, dist_tp) < self.distance

def find_base_distance_teleporter(self):
    cur = self.board_bot.position
    tp_in, tp_out, _ = self.find_nearest_teleport()
    if not tp_in or not tp_out:
        return float("inf")

```

```

        base = self.board_bot.properties.base
        return (abs(tp_in.x - cur.x) + abs(tp_in.y - cur.y) +
                abs(tp_out.x - base.x) + abs(tp_out.y - base.y))

    def find_nearest_diamond(self):
        direct_dist, direct_pos = self.find_nearest_diamond_direct()
        tp_dist, tp_path, tp_obj = self.find_nearest_diamond_teleport()
        red_dist, red_pos = self.find_nearest_red_button()

        if direct_dist <= tp_dist and direct_dist <= red_dist:
            self.static_goals = [direct_pos]
            self.distance = direct_dist
        elif tp_dist <= red_dist:
            self.static_goals = tp_path
            self.static_goal_teleport = tp_obj
            self.distance = tp_dist
        else:
            self.static_goals = [red_pos]
            self.distance = red_dist

    def find_nearest_red_button(self):
        cur = self.board_bot.position
        btn = self.redButton[0]
        dist = abs(btn.position.x - cur.x) + abs(btn.position.y - cur.y)
        return dist, btn.position

    def find_nearest_teleport(self):
        cur = self.board_bot.position
        min_dist = float("inf")
        best_tp, tp_in, tp_out = None, None, None
        for tp in self.teleporter:
            dist = abs(tp.position.x - cur.x) + abs(tp.position.y - cur.y)
            if dist and dist < min_dist:
                min_dist = dist
                best_tp = tp
                tp_in = tp.position
                tp_out = self.find_other_teleport(tp)
        return tp_in, tp_out, best_tp

    def find_other_teleport(self, tp: GameObject):
        return next((t.position for t in self.teleporter if t.id != tp.id),
None)

```

```

def find_nearest_diamond_teleport(self):
    cur = self.board_bot.position
    tp_in, tp_out, tp_obj = self.find_nearest_teleport()
    if not tp_in or not tp_out:
        return float("inf"), [], None

    best_diamond, min_dist = None, float("inf")
    for d in self.diamonds:
        if d.properties.points == 2 and self.board_bot.properties.diamonds
== 4:
            continue
        dist = (abs(tp_in.x - cur.x) + abs(tp_in.y - cur.y) +
                abs(d.position.x - tp_out.x) + abs(d.position.y -
tp_out.y)) / d.properties.points
        if dist < min_dist:
            min_dist = dist
            best_diamond = d
    if best_diamond:
        return min_dist, [tp_in, best_diamond.position], tp_obj
    return float("inf"), [], None

def find_nearest_diamond_direct(self):
    cur = self.board_bot.position
    best_diamond, min_dist = None, float("inf")
    for d in self.diamonds:
        if d.properties.points == 2 and self.board_bot.properties.diamonds
== 4:
            continue
        dist = (abs(d.position.x - cur.x) + abs(d.position.y - cur.y)) /
d.properties.points
        if dist < min_dist:
            min_dist = dist
            best_diamond = d
    return min_dist, best_diamond.position if best_diamond else None

def check_obstacle(self, obj_type):
    cur = self.board_bot.position
    dest = self.goal_position

    if obj_type == 'teleporter':
        objs = self.teleporter

```



```

elif obj_type == 'redDiamond':
    objs = [d for d in self.diamonds if d.properties.points == 2]
else:
    return

for o in objs:
    pos = o.position
    if pos == cur:
        continue
    if (pos.x == dest.x and min(cur.y, dest.y) < pos.y < max(cur.y,
dest.y)):
        alt_x = dest.x + 1 if dest.x <= 1 else dest.x - 1
        self.static_temp_goals = Position(dest.y, alt_x)
        self.goal_position = self.static_temp_goals
        return
    if (pos.y == dest.y and min(cur.x, dest.x) < pos.x < max(cur.x,
dest.x)):
        alt_y = dest.y + 1 if dest.y <= 1 else dest.y - 1
        self.static_temp_goals = Position(alt_y, dest.x)
        self.goal_position = self.static_temp_goals
        return

```

#### 4.1.2 Penjelasan Alur Program

Program ini bekerja dengan memanfaatkan algoritma greedy berbasis rasio point per jarak untuk menentukan pergerakan bot dalam permainan. Di awal, program akan melakukan inisialisasi beberapa variabel statis untuk menyimpan posisi tujuan, status teleporter, dan jarak antar objek. Pada setiap langkah permainan, fungsi `next_move()` akan dipanggil untuk menentukan arah gerak bot berdasarkan kondisi saat ini.

Bot akan memprioritaskan pengambilan diamond sebanyak mungkin dengan memilih diamond yang memberikan keuntungan paling tinggi relatif terhadap jaraknya. Jika kondisi tertentu terpenuhi, seperti jumlah diamond yang dibawa mencapai batas atau waktu hampir habis, bot akan langsung menuju base, baik melalui jalur biasa maupun menggunakan teleporter jika lebih efisien.

Proses pencarian diamond atau base dilakukan melalui beberapa fungsi seperti `find_nearest_diamond()`, `find_best_way_to_base()`, dan `find_nearest_teleport()`. Bot juga mempertimbangkan keberadaan objek penting seperti teleporter, red button, dan diamond berpoint 2. Selain itu, fungsi `check_obstacle()` digunakan untuk menghindari rintangan di jalur menuju

tujuan. Secara keseluruhan, program ini berjalan secara iteratif setiap giliran, menentukan posisi tujuan berikutnya, menghitung arah gerak, dan memutuskan apakah perlu menggunakan fitur teleport atau menghindari objek tertentu sebelum mencapai target.

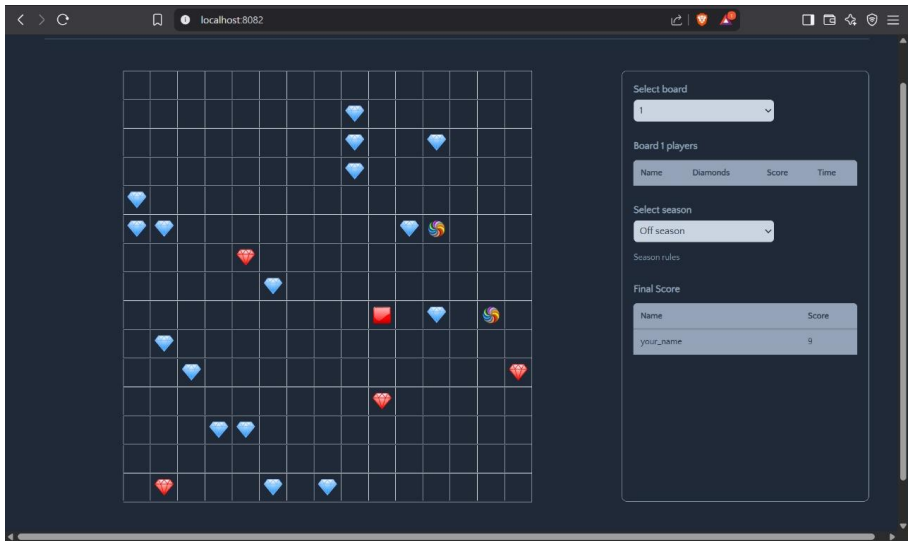
4.2 Struktur Data yang Digunakan

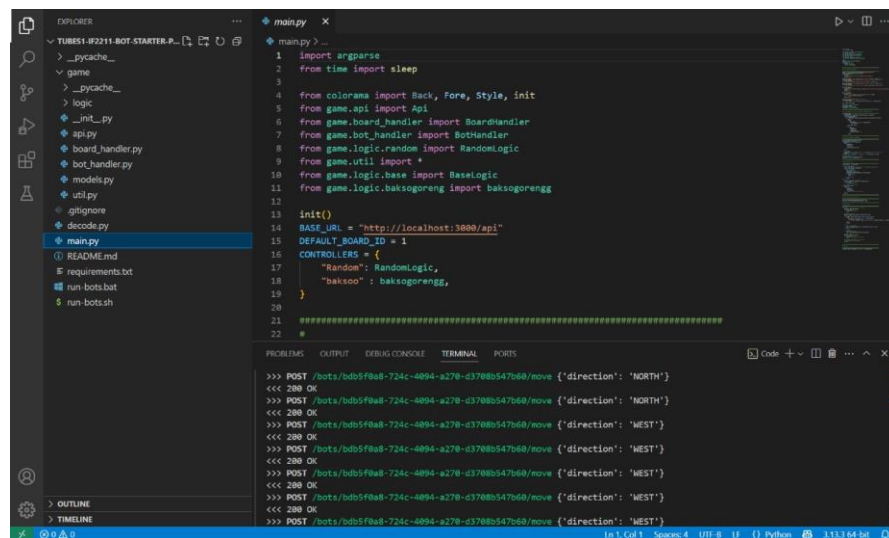
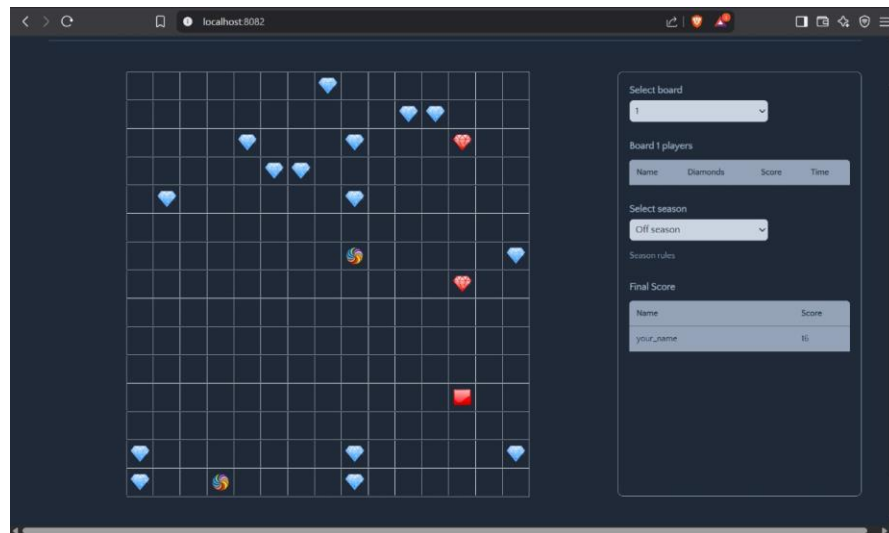
Struktur Data	Penggunaan di Pseudocode	Deskripsi
List	<code>static_goals: list[Position] = []</code>	Menyimpan daftar posisi tujuan (goal) secara berurutan.
	<code>self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]</code>	List tuple arah gerak (kanan, bawah, kiri, atas).
	<code>self.teleporter = [d for d in board.game_objects if d.type == "TeleportGameObject"]</code> <code>self.redButton = [d for d in board.game_objects if d.type == "DiamondButtonGameObject"]</code>	List berisi objek teleporter yang ditemukan di board.
Optional	<code>self.goal_position: Optional[Position] = None</code> <code>static_goal_teleport: Optional[GameObject] = None</code> <code>static_temp_goals: Optional[Position] = None</code>	Bisa berisi objek GameObject atau None.
Custom Object	<code>base_pos = Position(base.y, base.x)</code> <code>def next_move(self, board_bot: GameObject, board: Board):</code> <code>from game.models import Board, GameObject, Position</code>	Tipe data kustom dari game untuk menyimpan posisi (x, y) atau objek game lain.

4.3 Pengujian Program

4.3.1 Skenario Pengujian

Menjalankan bot lebih dari satu kali percobaan untuk melihat beberapa kemungkinan poin yang diperoleh seperti berikut :





### 4.3.2 Hasil Pengujian dan Analisis

Hasil pengujian, dari beberapa kali bot dijalankan, hasil yang didapatkan beragam dengan hasil paling banyak adalah 16 dan paling sering adalah 10, hal ini disebabkan kemunculan dari diamond dan base dari bot yang diacak sehingga hasilnya akan berbeda tergantung persebaran diamond dan lokasi basenya

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Dalam pengimplementasian algoritma greedy pada bot di permainan diamond terdapat alternatif solusi greedy yang kami gunakan. Namun dari berbagai alternatif yang telah dianalisis sebelumnya, strategi yang kami gunakan adalah greedy berdasarkan rasio nilai per Jarak yang artinya bot tidak hanya mempertimbangkan jarak terdekat semata, tetapi juga memperhitungkan nilai diamond yang akan diambil dibandingkan jarak yang harus ditempuh untuk mendapatkannya.

Bot ini memiliki beberapa kelebihan, antara lain kemampuannya dalam memilih rute berdasarkan densitas diamond secara efisien, serta strategi adaptif yang memungkinkannya langsung kembali ke base saat telah mengumpulkan banyak diamond atau waktu hampir habis. Selain itu, bot juga mampu memanfaatkan teleporter jika jalur tersebut lebih efisien dibandingkan rute langsung, sehingga dapat menghemat waktu. Namun, bot ini juga memiliki kelemahan, seperti kemungkinan terjebak karena tidak mempertimbangkan aksesibilitas jalur secara menyeluruh, serta mekanisme penghindaran rintangan yang sederhana, yang bisa menyebabkan bot berputar-putar tanpa keluar dari jebakan.

#### **5.2 Saran**

Bagi pembaca yang tertarik mengembangkan bot serupa, disarankan untuk mengeksplorasi lebih lanjut terkait pendekatan greedy dengan algoritma lain yang lebih kompleks dapat menghasilkan bot yang lebih cerdas, adaptif, dan kompetitif di berbagai skenario permainan. Selain itu, penting juga untuk mempertimbangkan strategi kompetitif, seperti mengantisipasi pergerakan bot lawan dan mengoptimalkan pemilihan target berdasarkan kondisi dinamis permainan.

## LAMPIRAN

A. Repository Github (<https://github.com/Sunnyior/Tubes-Stima-BaksoGoreng>)

B. Video Penjelasan

- Link Gdrive

(<https://drive.google.com/file/d/1SC3Ig4eRKmKlQsfc2NAYvJPBwrPtL43d/view?usp=drivesdk>)

- Link Youtube

(<https://youtu.be/m7A56nCHlQg?feature=shared>)

## DAFTAR PUSTAKA

- Strategi Pengoptimalan Kode Menggunakan Algoritma Greedy*. (2024, November 24). From Haltev.id: <https://haltev.id/strategi-pengoptimalan-kode-menggunakan-algoritma-greedy/>
- Chen, S. (2024, September 26). *Algoritma Greedy Beserta Contoh : Apa Itu, Metode dan Pendekatan*. From Guru99: <https://guru99.com/id/greedy-algorithm.html>