

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

## **Tugas Besar**

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



**Oleh: Kelompok KACANG KELEDAl.exe**

Muhammad Fauzan Naufal	123140150
Muhammad Fatahillah Farid	123140203
M. Reyshandi	123140037

Dosen Pengampu: Winda Yulita, M.Cs./Imam Eko Wicaksono, S.Si., M.Si.

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

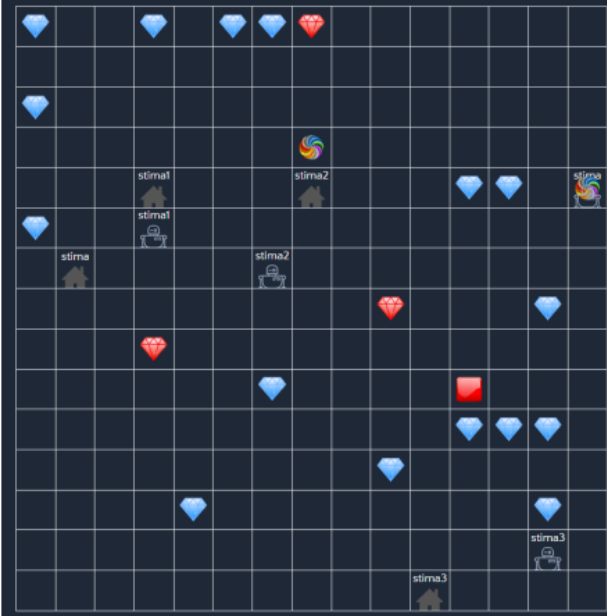
## DAFTAR ISI

<b>BAB I</b>	<b>DESKRIPSI TUGAS.....</b>	<b>3</b>
<b>BAB II.....</b>		<b>7</b>
<b>LANDASAN TEORI.....</b>		<b>7</b>
2.1 Dasar Teori.....		7
2.2 Cara Kerja Program.....		8
<b>BAB III.....</b>		<b>14</b>
<b>APLIKASI STRATEGI GREEDY.....</b>		<b>14</b>
3.1 Proses Mapping.....		14
3.2 Eksplorasi Alternatif Solusi Greedy.....		15
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy.....		16
3.4 Strategi Greedy yang Dipilih.....		16
4.1 Implementasi Algoritma Greedy.....		17
1. Pseudocode.....		17
2. Penjelasan Alur Program.....		18
4.2 Struktur Data yang Digunakan.....		18
4.3 Pengujian Program.....		19
1. Skenario Pengujian.....		19
2. Hasil Pengujian dan Analisis.....		21
<b>BAB V.....</b>		<b>22</b>
<b>KESIMPULAN DAN SARAN.....</b>		<b>22</b>
5.1 Kesimpulan.....		22
5.2 Saran.....		22
<b>LAMPIRAN.....</b>		<b>23</b>
<b>DAFTAR PUSTAKA.....</b>		<b>24</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.



The screenshot displays the Diamonds game interface. On the left is a 10x10 grid representing the game board. It contains several blue diamonds, red diamonds, and obstacles (represented by house and robot icons). Three player bots are visible, labeled stima1, stima2, and stima3. On the right is a control panel with the following sections:

- Select board:** A dropdown menu showing '1'.
- Board 1 players:** A table showing the status of the players.
- Select season:** A dropdown menu showing 'Off season'.
- Season rules:** A section for rules, currently empty.
- Final Score:** A table for the final scores.

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

Name	Score
------	-------

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

Program permainan Diamonds terdiri atas:

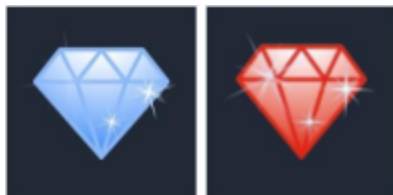
1. *Game engine*, yang secara umum berisi:
  - a. Kode *backend* permainan, yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *frontend* dan program bot
  - b. Kode *frontend* permainan, yang berfungsi untuk memvisualisasikan permainan
2. *Bot starter pack*, yang secara umum berisi:
  - a. Program untuk memanggil API yang tersedia pada *backend*
  - b. Program *bot logic* (bagian ini yang akan kalian implementasikan dengan algoritma *greedy* untuk bot kelompok kalian)
  - c. Program utama (*main*) dan utilitas lainnya

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

- ***Game engine*** :  
<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>
- ***Bot starter pack*** :  
<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

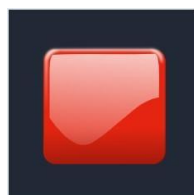
Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds



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

2. Red Button/Diamond Button



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

### 3. Teleporters



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

### 4. Bots and Bases



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

### 5. Inventory

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

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

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

1. Pertama, setiap pemain (bot) akan ditempatkan pada *board* secara *random*.  
Masing-masing bot akan mempunyai *home base*, serta memiliki *score* dan

*inventory* awal bernilai nol.

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

## **BAB II**

### **LANDASAN TEORI**

#### **2.1 Dasar Teori**

Algoritma greedy merupakan salah satu pendekatan dalam menyelesaikan masalah optimasi dengan cara melakukan pemilihan secara lokal terbaik pada setiap langkah dengan harapan akan menghasilkan solusi global yang optimal. Pendekatan ini bersifat iteratif, di mana pada setiap langkah algoritma memilih elemen yang tampak paling menguntungkan saat itu tanpa meninjau keseluruhan ruang solusi. Algoritma ini bekerja secara progresif, membangun solusi secara bertahap, dan setiap keputusan yang diambil bersifat final dan tidak dapat diubah di kemudian hari[1].

Agar algoritma greedy dapat memberikan solusi yang benar-benar optimal, suatu masalah harus memenuhi dua sifat utama, yaitu greedy choice property dan optimal substructure. Greedy choice property menyatakan bahwa pilihan terbaik yang diambil pada setiap langkah (lokal) akan menghasilkan solusi terbaik secara keseluruhan (global). Dengan kata lain, solusi optimal dapat diperoleh tanpa perlu meninjau semua kemungkinan kombinasi pilihan [1]. Sedangkan optimal substructure adalah sifat di mana solusi optimal dari suatu masalah dapat dibentuk dari solusi optimal dari submasalah-submasalahnya. Jika suatu masalah memiliki kedua sifat ini, maka algoritma greedy dapat digunakan secara efektif [2].

Keunggulan dari algoritma greedy terletak pada efisiensi waktu dan kesederhanaan implementasinya. Meskipun demikian, pendekatan ini tidak selalu menjamin solusi yang optimal untuk semua jenis masalah. Oleh karena itu, penting untuk menganalisis terlebih dahulu karakteristik dari permasalahan sebelum menerapkan algoritma ini [3]. Beberapa contoh penerapan algoritma greedy yang terkenal antara lain adalah masalah pemilihan aktivitas (activity selection), pembuatan pohon rentang minimum (minimum spanning tree) menggunakan algoritma Kruskal dan Prim, pengkodean Huffman (Huffman coding), serta masalah penukaran uang (coin change) dengan denominasi tertentu.

## 2.2 Cara Kerja Program

```
class KeledaiRakus(BaseLogic):
```

Baris kode `class KeledaiRakus(BaseLogic):` mendeklarasikan sebuah kelas baru dalam Python yang diberi nama `KeledaiRakus`. Nama ini mengindikasikan bahwa kelas tersebut kemungkinan besar akan mengimplementasikan suatu logika "rakus" atau Greedy. Bagian `(BaseLogic)` menunjukkan bahwa `KeledaiRakus` merupakan kelas turunan yang mewarisi sifat dan metode dari kelas induk `BaseLogic`, memungkinkannya untuk menggunakan fungsionalitas dasar yang sudah ada sambil menambahkan atau memodifikasi perilaku spesifiknya sesuai dengan pendekatan Greedy.

```
def __init__(self):
    self.goal_position: Optional[Position] = None
```

Potongan kode `def __init__(self): self.goal_position: Optional[Position] = None` adalah konstruktor untuk sebuah kelas dalam Python. Saat sebuah objek dari kelas ini dibuat, metode `__init__` ini otomatis dijalankan. Di dalamnya, ia menginisialisasi sebuah atribut instance bernama `goal_position` dengan nilai awal `None`. Petunjuk tipe `Optional[Position]` menandakan bahwa atribut `goal_position` ini diharapkan akan menyimpan objek bertipe `Position` atau bisa juga bernilai `None`, yang menunjukkan bahwa posisi tujuan mungkin belum ditentukan saat objek pertama kali dibuat.

```
def manhattan_distance(self, x1: int, y1: int, x2: int, y2: int) -> int:
    return abs(x1 - x2) + abs(y1 - y2)
```

Kode tersebut mendefinisikan sebuah fungsi bernama `manhattan_distance` di dalam sebuah kelas. Fungsi ini menghitung jarak Manhattan antara dua titik pada grid 2D, yaitu titik  $(x1, y1)$  dan  $(x2, y2)$ . Perhitungannya adalah jumlah dari selisih absolut koordinat x dan selisih absolut koordinat y.

```
# parameter avoid_red untuk menghindari diamond merah

def find_nearest_diamond(self, bot_position: Position, board: Board,
avoid_red: bool = False) -> Optional[Position]:

    diamonds = [

        obj for obj in board.game_objects

        if obj.type == "DiamondGameObject"

        and not (avoid_red and obj.properties.points == 2) # hindari
merah jika diminta
```



```

]

if not diamonds:

    return None

best_diamond = None

min_score = float('inf')

for diamond in diamonds:

    distance = self.manhattan_distance(bot_position.x,
bot_position.y, diamond.position.x, diamond.position.y)

    diamond_value = 2 if diamond.properties.points == 2 else 1

    score = distance / diamond_value

    if score < min_score:

        min_score = score

        best_diamond = diamond.position

return best_diamond

```

Fungsi `find_nearest_diamond` ini mencari posisi berlian terbaik di papan, dengan memperhitungkan posisi bot dan sebuah opsi untuk menghindari berlian merah (`avoid_red`). Ia pertama-tama menyaring daftar berlian sesuai kriteria tersebut. Kemudian, untuk setiap berlian yang valid, fungsi menghitung skor berdasarkan jarak dan nilai poinnya, di mana skor yang lebih rendah dianggap lebih baik. Akhirnya, ia mengembalikan posisi berlian dengan skor terbaik yang ditemukan, atau `None` jika tidak ada berlian yang memenuhi syarat.

```

def is_safe_move(self, next_x: int, next_y: int, board: Board) -> bool:

    for obj in board.game_objects:

        if obj.type == "BotGameObject":

            distance = abs(obj.position.x - next_x) +
abs(obj.position.y - next_y)

```

```

        if distance <= 2:

            return False

    return True

```

Fungsi `is_safe_move` ini memeriksa apakah langkah ke posisi tujuan (`next_x`, `next_y`) di papan (`board`) aman dari bot lain. Ia akan melihat semua objek di papan; jika ada objek lain yang merupakan "BotGameObject", fungsi ini menghitung jarak Manhattan ke bot tersebut dari posisi tujuan. Apabila jaraknya 2 unit atau kurang, langkah itu dianggap tidak aman dan fungsi mengembalikan `False`. Jika tidak ada bot lain dalam jarak berbahaya, fungsi mengembalikan `True`.

```

def next_move(self, board_bot: GameObject, board: Board) -> Tuple[int,
int]:

    current_position = board_bot.position

    props = board_bot.properties

    # Jika diamond penuh, langsung pulang

    if props.diamonds >= 5:

        self.goal_position = props.base

    # Jika diamond = 4, hindari diamond merah agar tidak overload

    elif props.diamonds >= 4:

        base_distance = self.manhattan_distance(current_position.x,
current_position.y, props.base.x, props.base.y)

        nearest_diamond = self.find_nearest_diamond(current_position,
board, avoid_red=True) # 🚫 hindari merah

        diamond_distance = self.manhattan_distance(

            current_position.x, current_position.y,

            nearest_diamond.x, nearest_diamond.y

```

```

        ) if nearest_diamond else float('inf')

        self.goal_position = props.base if base_distance <
diamond_distance else nearest_diamond

        # Jika diamond < 4, cari diamond terbaik (boleh merah)

        else:

                                                    self.goal_position =
self.find_nearest_diamond(current_position, board)

```

Fungsi `next_move` ini mengatur tujuan pergerakan bot (`self.goal_position`) berdasarkan jumlah berlian yang dipegang. Jika inventaris penuh (5 berlian atau lebih), bot akan langsung menargetkan markas. Apabila membawa tepat 4 berlian, ia akan memilih antara markas atau berlian non-merah terdekat, tergantung mana yang lebih dekat untuk dijangkau. Sedangkan jika berlian yang dibawa masih kurang dari 4, bot akan mencari dan menargetkan berlian terdekat jenis apapun sebagai tujuannya.

```

# Fallback jika tetap tidak ada target

if not self.goal_position:

    self.goal_position = props.base

```

Kode ini berfungsi sebagai pengaman atau fallback. Jika setelah semua logika penentuan target sebelumnya ternyata `self.goal_position` masih belum juga terisi (artinya bot tidak menemukan berlian atau tidak memenuhi syarat untuk pulang ke markas), maka baris ini akan memastikan bot tetap memiliki tujuan dengan mengatur `self.goal_position` ke markas (`props.base`). Ini mencegah bot menjadi tanpa arah.

```

# Hitung arah ke target

delta_x, delta_y = get_direction(

    current_position.x, current_position.y,

    self.goal_position.x, self.goal_position.y

)

```

```
next_x, next_y = current_position.x + delta_x, current_position.y + delta_y
```

Kode ini pertama-tama memanggil fungsi `get_direction` untuk menentukan arah langkah (`delta_x`, `delta_y`) dari posisi bot saat ini (`current_position`) menuju ke posisi tujuan (`self.goal_position`) yang telah ditetapkan. Setelah mendapatkan arah langkah tersebut, ia kemudian menghitung koordinat pasti (`next_x`, `next_y`) untuk posisi berikutnya dengan menambahkan `delta_x` dan `delta_y` ke koordinat posisi saat ini.

```
# Jika langkah tidak aman, cari alternatif aman yang tetap mendekati goal

if not self.is_safe_move(next_x, next_y, board):

    directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]

    safe_moves = []

    for dx, dy in directions:

        nx, ny = current_position.x + dx, current_position.y + dy

        if 0 <= nx < board.width and 0 <= ny < board.height and self.is_safe_move(nx, ny, board):

            distance_to_goal = self.manhattan_distance(nx, ny, self.goal_position.x, self.goal_position.y)

            safe_moves.append(((dx, dy), distance_to_goal))

    if safe_moves:

        safe_moves.sort(key=lambda item: item[1])

        return safe_moves[0][0]

# Jika semua arah bahaya, tetap maju ke goal

return delta_x, delta_y
```

```
return delta_x, delta_y
```

Setelah menentukan arah (`delta_x`, `delta_y`) menuju `goal_position`, kode ini pertama-tama memeriksa apakah langkah tersebut aman. Jika ternyata tidak aman, program akan mencari alternatif langkah lain ke segala arah mata angin (kanan, bawah, kiri, atas) yang aman dan masih berada dalam batas papan. Untuk setiap alternatif aman yang ditemukan, ia menghitung jaraknya ke `goal_position` yang sebenarnya. Dari semua opsi aman tersebut, program akan memilih arah yang paling mendekatkan bot ke tujuannya. Namun, jika setelah semua pengecekan tidak ada satupun langkah alternatif yang aman, bot akan tetap mengambil risiko dengan bergerak sesuai arah `delta_x`, `delta_y` awal. Jika langkah awal sudah dipastikan aman, maka arah tersebutlah yang akan langsung digunakan.

## **BAB III**

### **APLIKASI STRATEGI GREEDY**

#### **3.1 Proses Mapping**

Strategy greedy yang diterapkan pada permainan bot diamond kali ini memetakan permasalahan yang ada menjadi elemen elemen berikut:

##### **1. Himpunan Kandidat**

Merupakan seluruh kemungkinan yang dapat dijalankan dan akan dipilih solusinya berdasarkan kriteria tertentu. Pada program yang telah kami buat, kami memiliki 2 himpunan kandidat yaitu seluruh diamond yang ada dan arah dari pergerakan bot.

##### **2. Himpunan Solusi**

Merupakan kemungkinan solusi yang dapat dipilih berdasarkan himpunan kandidat berdasarkan kriteria yang telah ditetapkan. Pada program kami, himpunan solusinya adalah diamond yang memiliki nilai heuristik terbaik, serta arah pergerakan

##### **3. Fungsi Solusi**

Merupakan fungsi untuk menentukan tindakan terbaik yang dapat diambil berdasarkan kondisi saat ini. Dalam konteks permainan bot diamond, fungsi solusi adalah logika program yang menentukan keputusan yang harus diambil agar tujuan dapat tercapai (mengambil dan menyimpan diamond ke base). Adapun fungsi solusinya adalah:

- A. Find\_nearest diamond: Mengevaluasi dan memilih diamond yang diambil berdasarkan nilai heuristik terkecil (jarak/nilai).
- B. Manhattan\_distance: Sebagai dasar evaluasi nilai heuristik, berfungsi untuk mencari jarak terbaik untuk mencapai diamond terdekat.
- C. Is\_safe\_move: Memverifikasi langkah yang diambil, apakah aman dari jarak bot lain atau tidak, disini parameternya adalah  $\leq 2$  jarak manhattan.

Jika jarak dengan bot lain terlalu dekat, maka bot akan mengambil jalan alternatif.

- D. `Next_move`: Merupakan fungsi solusi utama yang berperan untuk melakukan pengambilan keputusan berdasarkan fungsi sebelumnya. Disini program juga menentukan goal set yang akan dilakukan berdasarkan kapasitas inventory pada bot, dimana jika `diamond == 5`, maka bot akan langsung pulang ke base, namun jika `diamond >= 4`, bot akan mempertimbangkan untuk mengambil diamond lagi atau pulang ke base dengan membandingkan jarak manhattan keduanya.

#### 4. Fungsi Objektif

Merupakan tujuan akhir dalam algoritma greedy, yaitu memaksimalkan nilai yang didapat pada rangkaian solusi yang dipilih. Pada program yang kami buat, adapun fungsi objektifnya yaitu:

- A. Memprioritaskan diamond dengan value paling baik (nilai heuristik terkecil).
- B. Menghindari bot lawan ketika jarak manhattannya  $\leq 2$  untuk meminimalisir resiko ditackle.
- C. Goal set kondisional berdasarkan kapasitas inventory, dimana jika `inventory == 5`, maka bot akan langsung menuju base, lalu jika `inventory >= 4`, bot akan membandingkan nilai heuristik berdasarkan jarak manhattan antara base dengan diamond terdekat untuk pengambilan keputusan apakah bot akan menuju base atau mengambil diamond.

### 3.2 Eksplorasi Alternatif Solusi Greedy

Alternatif solusi greedy yang terpikirkan kami sebelumnya adalah algoritma greedy berdasarkan jarak terdekat dengan diamond. Strategi ini hanya berfokus pada diamond dengan jarak manhattan paling dekat dengan bot tanpa mempedulikan nilainya.

### 3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Untuk mengevaluasi solusi greedy yang telah disebutkan sebelumnya, kami akan membandingkan tingkat efisiensi dan efektifitas solusi terhadap perilaku bot dalam game bot diamond.

Solusi greedy yang diimplementasikan dalam program adalah *Greedy by Value* dengan cara mempertimbangkan nilai heuristik untuk pengambilan diamond dan keputusan goal set berdasarkan kapasitas inventory. Solusi ini dinilai cukup efisien dan efektif karena memungkinkan bot untuk membuat keputusan yang cerdas dan dinamis, karena selain mengevaluasi jalan untuk mendapatkan nilai tertinggi, bot juga mempertimbangkan resiko dengan melakukan kalkulasi jarak aman dengan bot musuh.

Untuk solusi alternatif sebelumnya, yaitu strategi greedy yang hanya berdasarkan jarak terdekat dengan diamond, bot ini dinilai memiliki efisiensi yang sangat bagus karena hanya membutuhkan perhitungan jarak antar bot dengan diamond terdekat, jadi solusi ini sangat ringan secara komputasi dan cocok untuk pengambilan keputusan jangka pendek. Namun, efektivitas dengan solusi ini dinilai buruk karena diamond yang digenerate bersifat random, jadi ketika bot spawn dengan jarak yang cenderung tidak menguntungkan, bisa saja bot hanya mengambil diamond yang bernilai rendah secara terus menerus. Hal ini diperburuk dalam skala turnamen dimana efektivitas dalam pengambilan nilai menjadi sangat krusial.

### 3.4 Strategi Greedy yang Dipilih

Untuk strategi greedy yang kami pilih, disini kami menggunakan *Greedy by Value* dimana kami mengutamakan diamond dengan value yang paling besar (diamond merah). Penilaian ini berdasarkan jarak antar bot dengan diamond itu sendiri, yang dilakukan dengan fungsi heuristik antara jarak/nilai diamond. Selanjutnya, untuk menghindari resiko kehilangan inventory, bot kami menggunakan strategi defensif yaitu dengan mengkalkulasi jarak aman terhadap bot lawan sejauh  $< 2$  jarak manhattan. Lalu yang terakhir, bot akan mempertimbangkan inventory sebagai goal set (di luar batas maksimal 5) yaitu ketika inventory sudah mencapai  $\geq 4$ , bot akan membandingkan jarak antar diamond dengan base, perbandingan ini lagi lagi dilakukan dengan fungsi heuristi



## BAB IV

### IMPLEMENTASI DAN PENGUJIAN

#### 4.1 Implementasi Algoritma Greedy

##### 1. Pseudocode

```
CLASS KeledaiRakus EXTENDS BaseLogic
```

```
  INIT:
```

```
    goal_position = None
```

```
  FUNCTION manhattan_distance(x1, y1, x2, y2):
```

```
    RETURN |x1 - x2| + |y1 - y2|
```

```
  FUNCTION find_nearest_diamond(bot_position, board, avoid_red=False):
```

```
    diamonds = all diamond objects on board
```

```
    IF avoid_red:
```

```
      diamonds = remove red diamonds (points == 2)
```

```
    IF no diamonds:
```

```
      RETURN None
```

```
    FOR each diamond in diamonds:
```

```
      distance = manhattan_distance(bot_position, diamond)
```

```
      value = 2 if diamond is red else 1
```

```
      score = distance / value
```

```
      IF score < best score:
```

```
        best_diamond = diamond
```

```
    RETURN best_diamond position
```

```
  FUNCTION is_safe_move(next_x, next_y, board):
```

```
    FOR each bot on board:
```

```
      IF distance to (next_x, next_y) <= 2:
```

```
        RETURN False
```

```
    RETURN True
```

```
  FUNCTION next_move(board_bot, board):
```

```
    current_position = board_bot.position
```

```
    properties = board_bot.properties
```

```
    IF diamonds collected >= 5:
```

```
      goal = base
```

```
    ELSE IF diamonds collected == 4:
```

```
      base_dist = distance to base
```

```
      nearest_diamond = find_nearest_diamond(avoid_red=True)
```

```
      diamond_dist = distance to nearest_diamond
```

```

goal = base IF base_dist < diamond_dist ELSE nearest_diamond

ELSE:
    goal = find_nearest_diamond()

IF goal is None:
    goal = base

direction = get_direction(current_position, goal)
next_pos = current_position + direction

IF not is_safe_move(next_pos):
    FOR each possible direction:
        new_pos = current_position + direction
        IF in bounds AND is_safe_move:
            record safe move and distance to goal
    IF any safe move:
        RETURN move with shortest distance to goal
    ELSE:
        RETURN direction toward goal anyway

RETURN direction toward goal

```

## 2. Penjelasan Alur Program

- a. Inisialisasi bot
- b. Fungsi *manhattan\_distance*
- c. Fungsi *find\_nearest\_diamond*
- d. Fungsi *is\_safe\_move*
- e. Fungsi *next\_move*
- f. Menentukan tujuan
- g. Menghitung arah
- h. Validasi apakah jalur aman

## 4.2 Struktur Data yang Digunakan

Adapun struktur data yang digunakan dalam program kami yang meliputi:

- Class
  - GameObject: berupa semua objek yang ada pada game, yaitu bot, diamond, teleporter, red button.

-Board: menyimpan semua informasi permainan pada board.

-Position: menyimpan kordinat objek pada board (x, y).

- Tipe Data Built-in

-List: digunakan saat mengumpulkan objek yang dimaksud, penggunaanya dalam program adalah ketika mencari diamond terdekat (`diamonds = [obj for obj in board.game_objects if obj.type == "DiamondGameObject"]`).

-Optional: digunakan untuk menyatakan apabila suatu nilai bisa saja tidak diisi apa apa tergantung kondisi permainan, penggunaanya ada dalam (`self.goal_position: Optional[Position] = None`).

-Tuple: digunakan untuk menyimpan gerakan langkah yang ada pada fungsi `next_move` yang bernilai tetap.

-Float: digunakan untuk menyimpan nilai `min_distance` yang sangat besar (`inf`) agar diamond pertama dapat menjadi pembanding untuk dibandingkan dengan diamond berikutnya.

## 4.3 Pengujian Program

### 1. Skenario Pengujian

Disini kami melakukan 4 kali pengujian program dengan keterangan:

- Stima: menggunakan algoritma *greedy by value* yang kami kembangkan
- Stima1: menggunakan algoritma greedy berdasarkan posisi diamond terdekat.
- Stima2 & Stima 3: menggunakan algoritma random

Final Score	
Name	Score
stima1	12
stima	11
stima3	5
stima2	0

Final Score	
Name	Score
stima	10
stima1	7
stima3	3
stima2	0

Final Score	
Name	Score
stima	10
stima3	0
stima2	0
stima1	0

Final Score	
Name	Score
stima	15
stima1	6
stima2	1
stima3	0

## **2. Hasil Pengujian dan Analisis**

Berdasarkan hasil pengujian yang telah dilakukan, algoritma yang kami implementasikan dalam bot “stima” memiliki rasio 3:1. Kemenangan ini membuktikan algoritma yang kami implementasikan mampu bersaing secara kompetitif. Namun pada kondisi tertentu, solusi algoritma greedy sederhana (berdasarkan jarak diamond terdekat) memiliki keunggulan sendiri, dibuktikan pada pengujian 1 dimana algoritma yang kami rancang mengalami kekalahan tipis.

Secara umum, penggunaan algoritma greedy yang kami implementasikan untuk pemetaan board yang memiliki jarak diamond tidak terlalu padat dinilai efektif, namun masalah mulai muncul saat persaingan di daerah padat, dimana bot kami yang bersifat defensif cenderung kerepotan untuk menghindari serangan dari bot lain dan disaat bersamaan harus mengoleksi diamond untuk bersaing secara kompetitif.

## **BAB V**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Bot ini kami rancang dengan prinsip “rakus tapi berhitung”, yaitu terus mengumpulkan diamond sebanyak mungkin sambil mempertimbangkan jarak dan nilai untuk memaksimalkan efisiensi pergerakan. Meskipun demikian, bot tetap menghindari risiko dengan tidak melangkah terlalu dekat ke bot lawan ( $\text{radius} \leq 2 \text{ blok}$ ). Bot ini juga dapat menyesuaikan terhadap kondisi permainan, seperti otomatis kembali ke base saat penuh, menghindari diamond merah saat hampir penuh, dan mencari jalur alternatif jika terhalang. Prioritas yang diambil bersifat dinamis, menyesuaikan antara keamanan dan potensi keuntungan. Selain itu, sistem fallback disiapkan agar bot tidak macet dan tetap bisa kembali ke base jika tidak menemukan target yang layak.

#### **5.2 Saran**

Saran pengembangan dari bot ini untuk menambahkan kemampuan memprediksi pergerakan lawan agar dapat menghindari bot lain lebih cepat dan efisien, menerapkan kalkulasi pergerakan berdasarkan waktu atau strategi akhir permainan, memprioritaskan diamond terdekat saat waktu terbatas, mengoptimalkan perhitungan nilai diamond dengan mempertimbangkan faktor risiko serta ancaman dari bot lain, dan memperbaiki algoritma pemilihan rute agar lebih efisien dengan pemilihan jalur yang lebih aman.

## **LAMPIRAN**

**A. Repository Github [https://github.com/fawujann/Tubes1\\_KACANGKELEDAI](https://github.com/fawujann/Tubes1_KACANGKELEDAI)**

**B. Video Penjelasan (link GDrive)**

## DAFTAR PUSTAKA

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.
- [2] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Boston, MA: Addison-Wesley, 2011.
- [3] J. Kleinberg and É. Tardos, *Algorithm Design*. Boston, MA: Pearson, 2006.