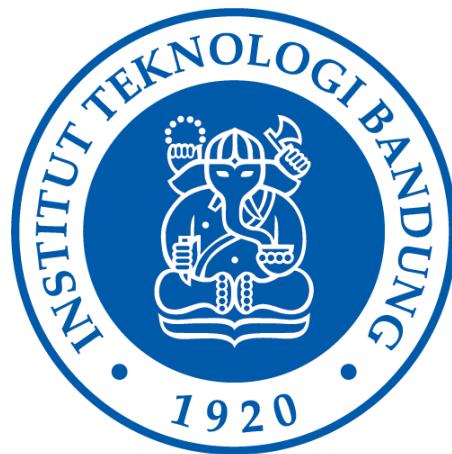


# LAPORAN TUGAS BESAR

## STRATEGI ALGORITMA IF2211

Pemanfaatan Algoritma Greedy dalam pembuatan logic untuk bot  
permainan Diamonds



**Disusun oleh Kelompok MrNaginski**

13522066 - Nyoman Ganadipa Narayana

13522084 - Dhafin Fawwaz Ikramullah

13522089 - Abdul Rafi Radityo Hutomo

## **Daftar Isi**

<b>Daftar Isi.....</b>	<b>2</b>
<b>BAB I Deskripsi Masalah.....</b>	<b>3</b>
<b>BAB II Landasan Teori.....</b>	<b>4</b>
A. Algoritma Greedy.....	4
B. Cara Kerja Program.....	4
<b>BAB III Aplikasi Strategi Greedy.....</b>	<b>6</b>
A. Strategy Greedy pada Bot Permainan Diamonds.....	6
B. Berbagai Alternatif Solusi.....	6
C. Solusi Pilihan.....	7
<b>BAB IV Implementasi dan Pengujian.....</b>	<b>8</b>
A. Penjelasan Algoritma.....	8
B. Struktur Data Algoritma.....	8
C. Analisis Solusi dan Pengujian.....	8
BAB V Kesimpulan dan Saran.....	10
<b>BAB VI Lampiran.....</b>	<b>11</b>
<b>BAB VII Daftar Pustaka.....</b>	<b>11</b>

## BAB I Deskripsi Masalah

Diamonds merupakan suatu programming challenge yang mempertandingkan bot buatan pemain. 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.

Pada game diamonds, permainan dilakukan di atas sebuah papan yang dikustomisasi ukurannya. Pada awal permainan, bot masing-masing pemain akan memulai permainan pada *base* masing-masing yang terletak pada suatu petak tertentu pada papan yang ditentukan secara *random*. Kemudian, pada papan terdapat beberapa diamond yang tersebar pada papan yang dapat dikumpulkan oleh pemain. Terdapat dua jenis diamond, yaitu diamond biru yang bernilai 1 poin dan diamond merah yang bernilai 2 poin. Poin yang didapat dari mengumpulkan diamond hanya akan dihitung ke dalam skor pemain ketika bot pemain tersebut telah mengambil diamond tersebut sehingga masuk ke dalam *inventory*-nya dan kembali ke *base* miliknya. Pada papan juga terdapat objek teleporter yang dapat dimasukkan oleh bot pemain untuk melakukan teleportasi ke teleporter yang berpasangan dengannya. Selain itu, juga ada tombol merah yang akan mengacak kembali diamond yang ada pada papan.

Untuk memenangkan game ini, pemain diperlukan untuk membuat sebuah bot yang mengimplementasikan algoritma tertentu. Bot yang dapat dibuat oleh pemain dapat melakukan gerakan sejauh 1 petak secara horizontal atau vertikal. Setiap bot memiliki *delay* antar gerakannya yang sama untuk seluruh pemain. Selain itu, bot pemain dapat melakukan *tackle* bot pemain lain dengan cara berpijak ke petak tempat bot itu berada. Ketika berhasil melakukan *tackle*, bot yang di-*tackle* akan terkirim kembali ke *base*-nya dan kehilangan seluruh diamond pada *inventory*-nya, diamond yang hilang tersebut akan berpindah ke *inventory* bot yang melakukan *tackle*.

Dengan ketentuan tersebut, perlu dibuat sebuah bot yang mengimplementasikan Algoritma Greedy untuk meraih potensi maksimum pada satu ronde permainan ini.

## **BAB II Landasan Teori**

### **A. Algoritma Greedy**

Algoritma Greedy merupakan algoritma yang memilih solusi terbaik berdasarkan informasi yang tersedia pada saat itu, tanpa mempertimbangkan dampak dari pilihan tersebut terhadap keputusan di masa depan. Keputusan yang diambil oleh algoritma ini didasarkan pada apa yang tampaknya optimal pada saat itu. Algoritma ini menunjukkan efisiensi yang tinggi dalam hal waktu eksekusi karena tidak perlu mengeksplorasi semua kemungkinan solusi.

Namun, perlu dicatat bahwa algoritma Greedy tidak selalu menghasilkan solusi optimal dalam konteks global. Dalam beberapa kasus, algoritma ini dapat menghasilkan solusi yang cukup baik, tetapi bukan yang terbaik. Sebagai contoh, dalam Travelling Salesman Problem, algoritma Greedy dapat memilih rute terpendek antara dua kota tanpa mempertimbangkan rute total, yang dapat menghasilkan rute yang bukan merupakan rute terpendek secara keseluruhan. Meski demikian, algoritma Greedy tetap memiliki kegunaan yang signifikan dalam situasi di mana mencari solusi optimal global tidak praktis atau membutuhkan terlalu banyak waktu dan sumber daya komputasi.

Dalam pengimplementasian algoritma greedy, ada elemen-elemen penting yang diperlukan untuk meraih hasil yang diinginkan.

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 : fungsi yang dimaksimumkan atau diminimumkan

### **B. Cara Kerja Program**

Diamonds merupakan permainan berbasis web, sehingga setiap aksi yang dilakukan – mulai dari mendaftarkan bot hingga menjalankan aksi bot – akan memerlukan HTTP request terhadap API endpoint tertentu yang disediakan oleh backend. Berikut adalah urutan requests yang terjadi dari awal mula permainan.

1. Program bot akan mengecek apakah bot sudah terdaftar atau belum, dengan mengirimkan POST request terhadap endpoint /api/bots/recover dengan body berisi email dan password bot. Jika bot sudah terdaftar, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut. Jika tidak, backend akan memberikan response code 404.
2. Jika bot belum terdaftar, maka program bot akan mengirimkan POST request terhadap endpoint /api/bots dengan body berisi email, name, password, dan team. Jika berhasil, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut.
3. Ketika id bot sudah diketahui, bot dapat bergabung ke board dengan mengirimkan POST request terhadap endpoint /api/bots/{id}/join dengan body berisi board id yang diinginkan (preferredBoardId). Apabila bot berhasil bergabung, maka backend akan memberikan response code 200 dengan body berisi informasi dari board.
4. Program bot akan mengkalkulasikan move selanjutnya secara berkala berdasarkan kondisi board yang diketahui, dan mengirimkan POST request terhadap endpoint /api/bots/{id}/move dengan body berisi direction yang akan ditempuh selanjutnya (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Apabila berhasil, maka backend akan memberikan response code 200 dengan body berisi kondisi board setelah move tersebut. Langkah ini dilakukan terus-menerus hingga waktu bot habis. Jika waktu bot habis, bot secara otomatis akan dikeluarkan dari board.
5. Program frontend secara periodik juga akan mengirimkan GET request terhadap endpoint /api/boards/{id} untuk mendapatkan kondisi board terbaru, sehingga tampilan board pada frontend akan selalu ter-update.

## **BAB III Aplikasi Strategi Greedy**

### **A. Strategy Greedy pada Bot Permainan Diamonds**

Pada pembuatan bot permainan diamonds, algoritma greedy dibuat dengan pertama-tama melakukan mapping dari elemen pada permainan diamonds kepada elemen pada algoritma greedy.

- Himpunan kandidat ( C ) : Tiles yang berada pada board
- Himpunan solusi, S : Tiles yang bersebelahan dengan posisi bot pada frame tersebut
- Fungsi solusi: Pemeriksaan apakah direction yang dipilih tidak out of bounds
- Fungsi seleksi (selection function): Memilih tiles yang meminimumkan fungsi objektif
- Fungsi kelayakan (feasible): Pemeriksaan apakah ada faktor lain yang perlu dipertimbangkan selain jarak rata-rata diamonds (inventory, waktu, dll)
- Fungsi objektif :  $f(tile) \rightarrow$  jarak rata-rata diamonds ke tile tersebut

### **B. Berbagai Alternatif Solusi**

#### **1. Naive**

Bot akan selalu mengejar diamond terdekat dari bot tersebut. Setelah waktu sudah mau habis, bot akan kembali ke base jika diamond ada di inventory.

#### **2. KodokSehat**

Bot akan selalu mengejar diamond terdekat yang tidak lebih dekat dari bot lain. Setelah waktu sudah mau habis, bot akan kembali ke base jika diamond ada di inventory. Setelah waktu sudah mau habis, bot akan kembali ke base jika diamond ada di inventory.

#### **3. KodokMahal**

Bot akan mengabaikan semua diamond yang terdekat dari bot lain. Lalu bot akan mengejar diamond terdekat dari sisanya. Setelah waktu sudah mau habis, bot akan kembali ke base jika diamond ada di inventory. Setelah waktu sudah mau habis, bot akan kembali ke base jika diamond ada di inventory. Strategi ini merupakan strategi yang mirip dengan KodokSehat, namun setelah dilakukan percobaan yang cukup banyak, KodokMahal lebih efisien dibanding KodokSehat.

#### **4. KodokPutih**

Strategi secara keseluruhan mengimplementasi gabungan algoritma *greedy* dan *complete search*. Apabila divisualisasikan, dikonstruksi sebuah graf, dengan diamond, bot, dan base dijadikan *node* dan setiap dua *node* dihubungkan dengan *edge* yang merepresentasikan jarak aktual mereka di board. Bot ini kemudian mengevaluasi dengan melakukan *search* urutan diamond apa saja yang akan ia kunjungi selama total jarak yang akan ia tempuh kurang dari 15. Fungsi evaluasi melibatkan urutan diamond yang sedang ia *search*, yaitu melalui 3 parameter: bobot *point* (modifikasi) keseluruhan pada urutan diamond, jarak total yang akan ditempuh untuk mengambil urutan diamond tersebut, dan jarak diamond terakhir yang ingin dicapai ke base. Dengan bobot *point* modifikasi suatu diamond yang digunakan bergantung pada jarak bot lain kepada diamond, yaitu apabila jarak bot lain lebih dekat kepada diamond tersebut, maka bobotnya tidak penuh, hanya 0.65 dari point awalnya. Evaluasi dari setiap urutan diamond dihitung dengan:

$$H = \frac{\text{bobot point}}{\text{total distance} \times \text{last diamond to base}}$$

Hasil evaluasi ini kemudian secara *greedy* dipilih yang terbesar. Kemudian tujuan dari bot di-set menjadi diamond pertama dalam urutan diamond tersebut. Setelah waktu sudah mau habis, bot akan kembali ke base.

## 5. VietCongRat

Bot akan menyeleksi diamond yang memiliki *manhattan distance* kurang dari sama dengan 9 petak dari posisi base, apabila tidak ada akan mengekspansi jarak menjadi 11 petak dari posisi base. Dari diamonds tersebut, bot akan mempertimbangkan 4 petak yang berhimpitan dengan petak posisi bot saat itu, bot akan memilih bot yang memiliki jarak rata-rata terhadap diamond yang sudah terseleksi bernilai minimum. Pengambilan diamond dilakukan ketika posisi bot berjarak kurang dari sama dengan 2 petak dari sebuah diamond, dimana bot akan bergerak menuju diamond tersebut. Selain itu, bot akan kembali ke base ketika *inventory* sudah nyaris penuh (kurang 1 diamond hingga penuh), atau ketika bot berjarak 2 dari base dan memiliki 3 atau lebih diamond pada inventory, atau ketika bot berjarak 1 dari base dan memiliki diamond dalam inventory. Setelah waktu sudah mau habis, bot akan kembali ke base jika diamond ada di inventory.

### **C. Solusi Pilihan**

Setelah dilakukan percobaan yang sangat banyak, ditemukan bahwa bot VietCongRat merupakan bot dengan strategy paling efisien. <vietcong stuff>

## BAB IV Implementasi dan Pengujian

### A. Penjelasan Algoritma

Algoritma dari VietCongRat <vietcong stuff>

Menggunakan struktur data program sebagai berikut,

```
TYPE Point:  
    < x: integer,  
      y: integer >  
  
TYPE Bot:  
    < name: string,  
      email: string,  
      id: string >  
  
TYPE Position: Point  
  
TYPE Base: Position  
  
TYPE Properties:  
    < points: integer OPTIONAL,  
      pair_id: string OPTIONAL,  
      diamonds: integer OPTIONAL,  
      score: integer OPTIONAL,  
      name: string OPTIONAL,  
      inventory_size: integer OPTIONAL,  
      can_tackle: boolean OPTIONAL,  
      milliseconds_left: integer OPTIONAL,  
      time_joined: string OPTIONAL,  
      base: Base OPTIONAL >  
  
TYPE GameObject:  
    < id: integer,  
      position: Position,  
      type: string,  
      properties: Properties OPTIONAL >  
  
TYPE Config:  
    < generation_ratio: float OPTIONAL,  
      min_ratio_for_generation: float OPTIONAL,  
      red_ratio: float OPTIONAL,  
      seconds: integer OPTIONAL,  
      pairs: integer OPTIONAL,
```

```

        inventory_size: integer OPTIONAL,
        can_tackle: boolean OPTIONAL >

TYPE Feature:
< name: string,
  config: Config OPTIONAL >

TYPE Board:
< id: integer,
  width: integer,
  height: integer,
  features: array of Feature,
  minimum_delay_between_moves: integer,
  game_objects: array of GameObject OPTIONAL,
  bots: array of GameObject,
  diamonds: array of GameObject >

TYPE VietCongRat:
< limit: integer,
  board: Board,
  my_bot: GameObject >

```

Pseudocode dari implementasi algoritma bot VietCongRat adalah sebagai berikut

#### VietCongRat.py

```

class VietCongRat extend BaseLogic {
KAMUS
    private limit: int
    private my_bot: GameObject
    private board: Board

IMPLEMENTASI
    constructor VietCongRat() {
        this.limit ← 9
    }
    public override function next_move(board_bot: GameObject,
board: Board) {
        KAMUS LOKAL
        constrained_diamonds: array of GameObject

```

```

    all_diamonds: array of GameObject
    diamonds_within_one: array of GameObject
    diamonds_within_two: array of GameObject
    diamonds_within_one: array of GameObject
    diamonds_within_two: array of GameObject

    ALGORITMA
        this.board ← board
        this.my_bot ← board_bot

            if (this.my_bot.properties.milliseconds_left//1000
<= distance_with_teleporter(this.my_bot.position,
this.my_bot.properties.base) + 2) then
                → move_toward_base()

                constrained_diamonds ←
get_all_diamonds_within_limit(this.limit)

                if(len(constrained_diamonds) == 0) then
                    constrained_diamonds ←
get_all_diamonds_within_limit(this.limit)

                if len(constrained_diamonds) == 2 then
                    →
move_towards_with_teleporter(min(constrained_diamonds, key=
lambda diamond: distance_with_teleporter(this.my_bot.position,
diamond.position)).position)

                all_diamonds ← this.board.diamonds
                diamonds_within_one ← this.diamonds_within_n(1,
all_diamonds)
                diamonds_within_two ← this.diamonds_within_n(2,
all_diamonds)
                diamonds_within_one ← list(filter(lambda diamond :
this.my_bot.properties.diamonds + diamond.properties.points <
this.my_bot.properties.inventory_size, diamonds_within_one))
                diamonds_within_two ← list(filter(lambda diamond :
this.my_bot.properties.diamonds + diamond.properties.points <
this.my_bot.properties.inventory_size, diamonds_within_two))

                if len(diamonds_within_one) > 0 then

                    for diamond in diamonds_within_one do
                        if (diamond.properties.points == 2) then
                            →

```

```

move_towards_with_teleporter(diamond.position)

    best_one_tile_away_diamond ←
min(diamonds_within_one, key=lambda diamond:
calculate_tile_avg_diamond_distance(diamond.position,
constrained_diamonds))
    →
move_towards_with_teleporter(best_one_tile_away_diamond.position)

    else if len(diamonds_within_two) > 0 then

        for diamond in diamonds_within_two do
            if (diamond.properties.points == 2) then
                →
move_towards_with_teleporter(diamond.position)

        best_two_tile_away_diamond ←
min(diamonds_within_two, key=lambda diamond:
calculate_tile_avg_diamond_distance(diamond.position,
constrained_diamonds))
        ←
move_towards_with_teleporter(best_two_tile_away_diamond.position)

        if (my_bot.properties.diamonds >=
my_bot.properties.inventory_size - 1) then
            → move_towards_base()
        else if (distance_with_teleporter(my_bot.position,
my_bot.properties.base) <= 2 and my_bot.properties.diamonds >=
3) then
            → move_towards_base()
        else if (distance_with_teleporter(my_bot.position,
my_bot.properties.base) <= 1 and my_bot.properties.diamonds >=
1) then
            → move_towards_base()

        best_avg_tile ←
calculate_tile_with_minimum_avg_diamond_distance_around_tile(my_
bot.position, constrained_diamonds)
        → move_towards_with_teleporter(best_avg_tile)
    }

private function get_both_teleporter() → (GameObject,
GameObject) {

```

```

        KAMUS LOKAL
            teleporters : array of GameObject
        ALGORITMA
            teleporters <- [d for d in board.game_objects
if d.type == "TeleportGameObject"]
                if len(teleporters) != 2 then
                    → (None, None)
                else then
                    → (teleporters[0], teleporters[1])
}
private function get_closer_tele() → GameObject {
    KAMUS LOKAL
        my_pos: Position
        tele1, tele2, closer_tele : GameObject
    ALGORITMA
        my_pos ← self.my_bot.position
        tele1, tele2 ← get_both_teleporter()

        if (self.distance(my_pos, tele1.position) >
self.distance(my_pos, tele2.position)) then closer_tele ← tele1
        else then closer_tele ← tele2

        → closer_tele
}

private function distance(a: Position, b: Position) → int {
    KAMUS LOKAL
        x, y: int
    ALGORITMA
        x ← abs(a.x - b.x)
        y ← abs(a.y - b.y)
        → x + y
}

private function distance_with_teleporter(a: Position, b:
Position) -> int {
    KAMUS LOKAL
        tele1, tele2, closer_tele : GameObject
    ALGORITMA
        tele1, tele2 ← get_both_teleporter()

        if (self.distance(a, tele1.position) <
self.distance(a, tele2.position)) then closer_tele ← tele1
        else then closer_tele ← tele2

```

```

        a_to_tele ← distance(a, closer_tele.position)
        tele_to_b ← distance(other_tele.position, b)
        classic_distance ← distance(a, b)

        → min(classic_distance, a_to_tele + tele_to_b)
    }

private function move_towards(dest: Position) → (int, int){

    KAMUS LOKAL
        delta_x, delta_y : int
        direction : (int, int)
        my_pos : Position
        out_of_bound_check : boolean
    ALGORITMA
        delta_x ← dest.x - this.my_bot.position.x
        delta_y ← dest.y - this.my_bot.position.y

        direction ← (0, 0)

        if abs(delta_x) > abs(delta_y) then
            direction ← (1 if delta_x > 0 else -1, 0)
        else then
            direction ← (0, 1 if delta_y > 0 else -1)

        my_pos ← self.my_bot.position

        out_of_bound_check = (my_pos.x + direction[0] >= 0
and my_pos.y + direction[1] >= 0 and my_pos.x +
direction[0] < self.board.width + 1 and my_pos.y +
direction[1] < self.board.height + 1)

        if (out_of_bound_check) then
            → direction
        else then
            → (direction[0]*(-1), direction[1]*(-1))
    }

    private function move_towards_with_teleporter(dest:
Position) → (int, int) {
        KAMUS LOKAL
            my_pos, closer_tele: Position
        ALGORITMA
            my_pos ← this.my_bot.position

```

```

        if (distance_with_teleporter(my_pos, dest) ==  

self.distance(my_pos, dest)) then  

            → self.move_towards(dest)  

        else if ((self.distance_with_teleporter(my_pos,  

dest) < self.distance(my_pos, dest))) then  

            closer_tele ← get_closer_tele()  

            → self.move_towards(closer_tele.position)  

    }  
  

private function move_towards_base() → (int, int) {  

    KAMUS LOKAL  

    ALGORITMA  

    →  

    move_towards_with_teleporter(my_bot.properties.base)
}  
  

private function move_towards_center() → (int, int) {  

    KAMUS LOKAL  

    ALGORITMA  

    →  

move_towards_with_teleporter(Position(board.height//2,  

board.width//2))
}  
  

private function get_all_enemy() → array of GameObject {  

    KAMUS LOKAL  

    enemy_bot_list: array of GameObject  

    ALGORITMA  

    enemy_bot_list <- self.board.bots  

remove(enemy_bot_list, self.my_bot)  

→ enemy_bot_list  

}
  

private function closest_enemy() → GameObject {  

    KAMUS LOKAL  

    curr_closest: GameObject  

    curr_closest_distance: int  

    enemies: array of GameObject
}
```

```

ALGORITMA
    curr_closest <- Null
    curr_closest_distance <- 30
    enemies = get_all_enemy()
    i traversal [0..enemies.length]
        if
            (distance_with_teleporter(self.my_bot.position,
            enemies[i].position) < curr_closest_distance) then
                curr_closest_distance <-
            distance_with_teleporter(self.my_bot.position,
            enemies[i].position)
                curr_closest <- enemies[i]
            -> curr_closest

    }

    private function get_bot_index(target_bot: GameObject) →
GameObject {
    KAMUS LOKAL
        bots: array of GameObject
        i: int
    ALGORITMA
        bots <- board.bots
        i <- 0
        while (i < len(bots) and not done) do
            if (bots[i] == target_bot) then
                done = true
            i++
        -> i

    }

    private function get_all_diamonds_within_limit(limit: int)
→ array of GameObject {
    KAMUS LOKAL
        diamonds: array of GameObject
        result: dynamic array of GameObject
        i: integer
    ALGORITMA
        diamonds <- board.diamonds
        i traversal [0..length(diamonds)]
            if (distance_with_teleporter(my_bot.position,
            diamonds[i].position) <= limit) then

```

```

        add(result, diamonds[i])
    -> result

}

private function calculate_tile_avg_diamond_distance(tile:
Position, diamonds : array of GameObject) -> float {
    KAMUS LOKAL
        total_distance: real
        i: integer
    ALGORITMA
        total_distance <- 0
        i traversal [0..length(diamonds)]
        total_distance <- total_distance +
    (distance_with_teleporter(tile, diamonds[i].position) *
    diamonds[i].properties.points * diamonds[i].properties.points)
    ^ 0.25
        -> total_distance / length(diamonds)

}
private function
calculate_tile_with_minimum_avg_diamond_distance_around_tile(ti
le: Position, diamonds : array of GameObject): Position {
    KAMUS LOKAL
        tiles: array of Position
        min_tile: Position
        min_distance: float
        current_distance: float
    ALGORITMA
        tiles = []
        add(tiles, Position(tile.y + 1, tile.x))
        add(tiles, Position(tile.y - 1, tile.x))
        add(tiles, Position(tile.y, tile.x + 1))
        add(tiles, Position(tile.y, tile.x - 1))

        min_tile <- tiles[0]
        min_distance <-
    calculate_tile_avg_diamond_distance(min_tile, diamonds)

        i traversal [1..tiles.length]
        current_distance =
    calculate_tile_avg_diamond_distance(tiles[i], diamonds)
        if (current_distance < min_distance) then
            min_distance = current_distance
            min_tile = tiles[i]

```

```

        -> min_tile

    }

    private function diamonds_within_n(n : int, diamonds: array
of GameObject) : Array of GameObject {
    KAMUS LOKAL
        d, i : int
        Neff: int
        result: dynamic array of GameObject
    ALGORITMA
        i traversal [0..diamond.length]
            if (distance_with_teleporter(my_bot.position,
diamond.position) <= n) then
                add(result, diamonds[i])
            -> result

    }
}

```

## B. Struktur Data Algoritma

Class VietCongRat memiliki atribut dan method sebagai berikut:

No.	Atribut	Tipe Data	Penjelasan
1.	limit	integer	Faktor pembatas
2.	board	Board	Menyimpan data board
3.	my_bot	GameObject	Menyimpan bot diri sendiri

No.	Fungsi	Penjelasan
1.	<u>__init__</u>	Konstruktor kelas

2.	next_move	Mengembalikan arah pergerakan berikutnya berdasarkan algoritma bot
3.	get_both_teleporter	Mengembalikan kedua GameObject teleporter yang ada pada board
4.	get_closer_tele	Mengembalikan GameObject teleporter yang berjarak lebih dekat terhadap bot
5.	distance	Mengembalikan nilai manhattan distance antara dua posisi
6.	distance_with_teleporter	Mengembalikan nilai manhattan distance antara dua posisi dengan mengambil yang lebih dekat antara dua kemungkinan (melalui teleporter dan tidak)
7.	move_towards	Mengembalikan arah direction untuk bergerak ke suatu posisi tanpa mempertimbangkan teleporter
8.	move_towards_with_teleporter	Mengembalikan arah direction untuk bergerak ke suatu posisi dengan mempertimbangkan jalur terdekat melalui teleporter
9.	move_towards_base	Mengembalikan arah direction untuk bergerak kembali ke base dengan mempertimbangkan jalur terdekat melalui teleporter
10.	move_towards_center	Mengembalikan arah direction untuk bergerak ke tengah board dengan mempertimbangkan jalur terdekat melalui teleporter
11.	get_all_enemy	Mengembalikan seluruh GameObject bot musuh
12.	closest_enemy	Mengembalikan sebuah GameObject, yaitu bot musuh yang terdekat
13.	get_bot_index	Menentukan indeks suatu bot pada array board.bots
14.	get_all_diamonds_within_limit	Mengembalikan seluruh diamond pada jangkauan limit bot

15.	calculate_tile_avg_diamond_distance	Menghitung nilai rata-rata dari akar pangkat 4 jarak seluruh diamond pada array diamonds ke suatu Position
16.	calculate_tile_with_minimum_avg_diamond_distance_around_tile	Mengembalikan petak yang memiliki nilai rata-rata jarak terdekat dari 4 petak yang berada di sekitar Position yang diinput
17.	diamonds_within_n	Mengembalikan seluruh diamond yang berjarak n dari bot

### C. Analisis Solusi dan Pengujian

Beberapa bot yang digunakan difilter terlebih dahulu. Lalu dari 3 bot terbaik beserta bot naive, akan dilakukan percobaan yang cukup banyak. Data hasil percobaan dapat dilihat pada data sebagai berikut.

Strategi	Jumlah menang
VietCongRat	14
KodokMahal	12
KodokPutih	10
Naive	8

Dari tabel tersebut dapat dilihat bahwa bot VietCongRat mendapatkan skor tertinggi. Tapi walaupun begitu, bot lain mendapatkan skor yang tidak jauh di bawah VietCongRat. Jika percobaan dilakukan lebih banyak lagi, maka bot lain memiliki kemungkinan untuk menang lebih banyak dari bot VietCongRat. Ditambah lagi walaupun bot Naive tidak menang sebanyak bot lain, jumlah kemenangannya sebenarnya juga relatif banyak. Hal ini dapat menunjukkan bahwa sebenarnya faktor keberuntungan memiliki pengaruh yang cukup besar. Faktor keberuntungan yang dimaksud diantaranya adalah lokasi base yang, lokasi diamond tersebar, urutan jalan, delay tiap langkah. Dilakukan pula percobaan dengan pengaturan 2 lawan 2 yaitu sebagai berikut.

KodokPutih vs KodokMahal		KodokPutih vs VietCongRat		KodokMahal vs VietCongRat	
KodokPutih	KodokMahal	KodokPutih	VietCongRat	KodokMahal	VietCongRat
7	7	12	2	6	8

Dari tabel tersebut dapat dilihat bahwa ada yang hasilnya cukup konsisten yaitu pada KodokPutih lawan VietCongRat. Namun untuk hasil yang lain karena jumlah menangnya hampir sama, jika kita menarik kesimpulan maka hasilnya akan kurang akurat. Jika percobaan ditambah lebih banyak lagi, terkadang bot yang menang lebih sedikit akan menjadi menang lebih banyak.

## **BAB V Kesimpulan dan Saran**

### **Kesimpulan**

Algoritma greedy merupakan algoritma berupa pemilihan solusi terbaik berdasarkan informasi keadaan saat itu tanpa mempertimbangkan konsekuensi ke depan. Algoritma ini akan memiliki efisiensi yang tinggi karena tidak perlu mengeksplorasi semua kemungkinan solusi. Salah satu aplikasi dari algoritma ini adalah pada permainan diamonds. Namun terdapat kekurangan dari sistem permainan yang membuat sebagian besar penentu kemenangan adalah dengan faktor keberuntungan. Salah satu strategi efisien yang ditemukan adalah dengan cara menghitung rata-rata jarak diamond yang memiliki radius tertentu dari base terhadap keempat petak yang tepat di sebelah posisi bot pada saat ini dan memilih petak yang memiliki rata-rata paling rendah.

### **Saran**

## **BAB VI Lampiran**

Source code: [https://github.com/ganadipa/Tubes1\\_Mr-Naginski](https://github.com/ganadipa/Tubes1_Mr-Naginski)

Link video: <https://youtu.be/9gj4bIl9nEw?si=4ymjyIKXNyhR0lPb>

## **BAB VII Daftar Pustaka**

Rinaldi, M. (n.d.). Algoritma Greedy (2024) Bag1.

[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)

Rinaldi, M. (n.d.). Algoritma Greedy (2024) Bag2.

[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)

Rinaldi, M. (n.d.). Algoritma Greedy (2024) Bag3.

[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)