

**LAPORAN TUGAS BESAR I IF2211 STRATEGI ALGORITMA  
SEMESTER II 2023/2024  
“PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT  
PERMAINAN DIAMONDS”**

Kelompok 27 / sambut-ramadhan

Ibrahim Ihsan Rasyid	13522018
Muhammad Syaraf Akmal	13522076
M Rifki Virziadeili Harisman	13522120



**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG 2023**

## **DAFTAR ISI**

<b>BAB I</b>	<b>3</b>
<b>BAB II</b>	<b>4</b>
<b>BAB III</b>	<b>5</b>
<b>BAB IV</b>	<b>8</b>
<b>BAB V</b>	<b>21</b>
<b>LAMPIRAN</b>	<b>22</b>
<b>DAFTAR PUSTAKA</b>	<b>23</b>

## **BAB I**

### **DESKRIPSI MASALAH**

Diamonds merupakan suatu programming challenge yang mempertandingkan bot yang dibuat 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.

Bahasa pemrograman yang digunakan pada tugas besar ini adalah Python. Permainan Diamonds ini berbasis web dan dikembangkan oleh Etimo. Kode game engine dan starter pack dari bot sudah disediakan, sehingga pengembang bot hanya perlu mengimplementasikan algoritma bot pada starter pack. Berikut adalah cuplikan layar dari papan permainan Diamonds

Dalam sebuah papan, terdapat beberapa komponen permainan sebagai berikut :

1. Diamond  
Diamond adalah elemen yang dikumpulkan oleh bot dengan melewati/melangkahinya. Ada 2 jenis diamond, yaitu diamond merah dan diamond biru. Diamond merah bernilai 2 poin, sedangkan diamond 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  
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. Teleporter  
Terdapat tepat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.
4. Bot dan Base  
Setiap bot memiliki sebuah Base yang 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  
Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum yaitu 5 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**

#### **A. Dasar Teori Algoritma Greedy**

Algoritma Greedy adalah salah satu algoritma yang memecahkan persoalan dengan pendekatan melalui serangkaian langkah yang dikomputasikan langkah demi langkah hingga solusi ditemukan. Pada setiap langkah, keputusan yang diambil adalah *feasible* (tidak melanggar batasan dari persoalan), *locally optimal* (langkah terbaik dari semua kemungkinan yang ada pada langkah tersebut), dan *irrevocable* (tidak bisa mundur untuk mengulangi langkah yang sudah dilakukan). Mudahnya, pada setiap langkah, algoritma mengambil pilihan terbaik yang ada saat itu dengan harapan bahwa pilihan tersebut mengarahkan ke solusi yang optimal secara global

Algoritma Greedy dibangun atas enam elemen, yaitu:

1. Himpunan kandidat, berisi kandidat yang akan dipilih pada setiap langkah
2. Himpunan solusi, berisi kandidat yang sudah dipilih
3. Fungsi solusi, menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi seleksi, memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik
5. Fungsi kelayakan, memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi
6. Fungsi objektif, memaksimumkan atau meminimumkan

Jika solusi terbaik mutlak tidak terlalu diperlukan, maka algoritma greedy lebih baik digunakan untuk menghasilkan solusi hampiran daripada menggunakan algoritma yang kebutuhan waktunya eksponensial untuk menemukan solusi yang eksak.

#### **B. Cara Kerja Program secara Umum**

Pada permainan Diamonds, tujuan dari bot adalah untuk memenangkan permainan dengan mengambil sebanyak-banyaknya diamond yang ada pada papan dan menyimpannya pada base. Karena permainan ini merupakan permainan berbasis web, maka setiap aksi yang dilakukan – mulai dari mendaftarkan bot hingga menjalankan aksi bot – akan menggunakan HTTP request terhadap API endpoint tertentu yang disediakan oleh backend.

Bot pada permainan Diamonds dapat dibuat dengan mengimplementasikan algoritma Greedy. Setiap langkah yang diambil oleh bot dikalkulasi dengan greedy berdasarkan suatu strategi tertentu, misal berdasarkan jarak diamond biru terdekat dari bot, jarak bot ke base-nya, dan lain-lain. Pertimbangan pada setiap langkah bisa saja tidak hanya satu, namun ada hal-hal yang bisa dipertimbangkan secara bersamaan.

## **BAB III**

### **STRATEGI ALGORITMA GREEDY**

#### **A. Elemen-elemen Algoritma Greedy pada Persoalan**

Pada implementasi algoritma Greedy, terdapat beberapa elemen yang dapat didefinisikan, yaitu himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Pada persoalan permainan Diamonds, elemen-elemen algoritma Greedy dapat didefinisikan berikut :

- Himpunan kandidat : Himpunan dari langkah yang bisa dilakukan bot
- Himpunan solusi : Himpunan dari pilihan langkah yang memungkinkan untuk dilakukan suatu waktu
- Fungsi solusi : Memeriksa apakah langkah yang dipilih sesuai dengan prioritas yang sudah ditentukan untuk dilakukan
- Fungsi seleksi : Memilih langkah yang menghasilkan keuntungan terbesar untuk meningkatkan kemungkinan menang
- Fungsi kelayakan : Memeriksa apakah langkah valid
- Fungsi objektif : Perolehan diamond total maksimum

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

Berdasarkan elemen-elemen yang sudah ditentukan, terdapat beberapa alternatif solusi dalam pendekatan menggunakan algoritma Greedy. Meskipun objektifnya jelas yaitu mendapatkan perolehan diamond terbanyak di antara bot lainnya, namun adanya beberapa komponen selain diamond yang dapat menentukan berjalannya permainan membuat kemungkinan solusi persoalan menjadi sangat beragam. Oleh karena itu, terdapat beberapa teknik atau strategi greedy yang dapat diimplementasikan pada bot ini sebagai berikut :

##### **1. Greedy by Ratio**

Greedy by ratio memprioritaskan nilai rasio dari objek diamond pada map. Rasio berupa perbandingan nilai point diamond terhadap jarak (point/distance). Objek diamond dengan rasio terbesar akan menjadi tujuan pergerakan dari bot.

##### **2. Greedy by Distance**

Greedy by distance memprioritaskan jarak diamond terdekat dari bot.

##### **3. Greedy by Point (Diamond)**

Greedy by point memprioritaskan point dari objek diamond pada map. Objek dengan nilai poin terbesar dan jarak terdekat akan menjadi tujuan dari pergerakan bot.

#### **4. Greedy by Attack (Tackle)**

Greedy by attack menerapkan algoritma tambahan berupa “*fight or flight*” dimana jika berpapasan dengan bot lain (dalam radius tertentu), current bot akan menyerang atau menghindari berdasarkan kriteria berupa jumlah diamond.

### **C. Analisis Efisiensi dan Efektivitas dari Alternatif Solusi**

Dari beberapa alternatif solusi yang telah dirumuskan pada poin B, perlu dilakukan analisis efisiensi dan efektivitas untuk menentukan strategi yang ingin diimplementasikan

#### **1. Analisis Strategi by Ratio**

Strategi ini memprioritaskan rasio perbandingan dari points yang dimiliki oleh diamond terhadap jarak, dengan rasio terdapat dua konsiderasi (jarak dan points) yang dapat disatukan saat menentukan tujuan diamond. Kelemahan algoritma ini adalah tidak memperhatikan kondisi sekitar dari diamond sehingga tidak optimum pada kasus spesifik seperti apabila terdapat jarak yang sama antara dua diamond yang berbeda nilai dimana daerah diamond yang lebih tinggi tidak memiliki/sedikit persebaran diamondnya dibandingkan diamond yang lebih rendah.

#### **2. Analisis Strategi by Distance**

Strategi ini memprioritaskan jarak terdekat diamond terhadap bot, dengan strategi ini pergerakan bot jadi lebih taktis. Kelemahan algoritma ini adalah tidak memperhatikan parameter nilai diamond sebagai faktor penentu. Hal ini berakibat dalam kondisi spesifik dimana saat skor perlu dua point untuk kembali ke base namun bot malah memilih diamond terdekat yang memiliki nilai lebih rendah sehingga algoritma tidak efektif.

#### **3. Analisis Strategi by Point**

Strategi ini memprioritaskan nilai dari diamond dan memilih jalur terdekat dari diamond tersebut, sehingga pendapatan nilai diamond lebih maksimum dan efektif. Kelemahan strategi ini adalah tidak taktis dan hanya memperhatikan diamond dengan nilai lebih tinggi saja sehingga algoritma tidak efisien.

#### **4. Analisis Strategi by Attack**

Strategi ini memprioritaskan penyerangan bot dengan kriteria yang dapat membuat diamond point penuh, sehingga tidak perlu mencari diamond di sekitar lebih lama dan meningkatkan kemungkinan untuk menang kompetisi dengan bot lain. Kelemahan algoritma ini adalah apabila bot musuh memiliki algoritma untuk

menghindar, maka akan terjadi kejar-kejaran yang justru membuat algoritma tidak efisien.

#### **5. Analisis Strategi by another game object**

Strategi ini mempertimbangkan pemilihan keputusan ketika bertemu dengan teleporter dan kondisi diamond sudah sedikit serta jauh. Pada kondisi dekat dengan teleporter, bot akan melakukan pengecekan goal apakah yang dia inginkan, jika mencari diamond maka akan melakukan pengecekan kembali. Setelah itu akan dibuat list rasio antara diamond terdekat dengan bot dan terdekat dengan teleporter. Setelah itu nilai yang paling maksimal akan menentukan kemana bot akan bergerak, menghindari teleporter atau menuju teleporter

Strategi yang berikutnya adalah mempertimbangkan red button ketika jumlah diamond sudah sedikit dan jarak nya cukup jauh dibandingkan jarak ke red button. Untuk implementasinya tersendiri kita tetap menggunakan list rasio diamond, jika list rasio diamond lebih kecil dari poin red button maka akan dipilih red button. Jika tidak maka sebaliknya. Untuk poin dari red button kita membuat rumus tersendiri untuk menghitungnya (bisa dilihat di dalam kode)

#### **D. Strategi Greedy yang Dipilih**

Berdasarkan analisis efisiensi dan efektivitas dari alternatif-alternatif solusi yang ada, kami memutuskan untuk mengkombinasikan strategi greedy by ratio, greedy by ratio in another object (red button and teleporter) dan by attack. Algoritma ini kami pilih karena dari perbandingan alternatif greedy by ratio, distance, dan point kami melihat bahwa greedy by ratio menggunakan point dan distance sebagai konsiderasi sehingga dapat membuat bot menjadi efektif dan efisien (walaupun tidak se-efektif greedy by point dan tidak se-efisien greedy by distance). Kami juga mengimplementasikan greedy by attack dengan konsiderasi sebagai antisipasi penyerangan dari bot musuh. Selain itu untuk menambah efektifitas pergerakan kita juga mengko

## BAB IV IMPLEMENTASI DAN PENGUJIAN

### A. Implementasi Algoritma Greedy

#### a. Implementasi method coordinate\_diamond\_ratio

Function coordinate\_diamond\_ratio(List[position]: coordinates, List[int]: points, int: current\_position) → List[position, int], List[float]

{ IS: list koordinat dari kumpulan objek diamond, list poin dari kumpulan objek diamond, posisi inisial yang diinginkan terdefinisi }

{ FS: akan menghasilkan list of list koordinat dan points, juga ratio dari setiap objek diamond yang didapat dari poin/jarak (jika jarak = 0 maka akan diassign -1) }

#### KAMUS

ratio : List[float]

#### ALGORITMA

```
// Inisialisasi list untuk menyimpan ratio dari setiap diamond
ratio = []
// Menghitung jumlah koordinat dari setiap diamond ke posisi saat ini
sum_coordinates ← [((coordinate.x - current_position.x)^2 +
                    (coordinate.y-current_position.y)^2) for coordinate in coordinates]

// Iterasi melalui setiap koordinat
i iterate [0... len(sum_coordinates)]:
    // Memeriksa apakah jarak antara koordinat dan posisi saat ini adalah 0
    if (sum_coordinates[i] = 0) then:
        // Jika jarak = 0, menetapkan ratio ke -1
        ratio.append(-1)
    else:
        // Menghitung dan menambahkan ratio untuk diamond ke dalam list ratio
        ratio.append(points[i]/sum_coordinates[i])

// Mengembalikan list koordinat, list poin, dan list ratio
→ [coordinates, points], ratio
```



## b. Implementasi method `get_coordinate_goal_for_diamond`

Function `get_coordinate_goal_for_diamond(List[float]: list_ratio, List[position]: list_coordinates, int inventory_diamonds, List[int]: list_point) → position, boolean`  
{ IS: list ratio diamond, list koordinat, jumlah diamond yang dibawa, dan list point terdefinisi }  
{ FS: akan menghasilkan koordinat dari diamond yang diinginkan }

### KAMUS

`max_ratio`: float  
`index`: int

### ALGORITMA

```
// Cari index dari nilai maksimum dalam list ratio
max_ratio ← max(list_ratio)
index ← list_ratio.index(max_ratio)

// Mengabaikan diamond berpoin 2 jika jumlah diamond yang dibawa adalah 4
while (list_point[index] = 2 and inventory_diamonds = 4) do:
    list_ratio[index] ← list_ratio[index] - 1
    max_ratio ← max(list_ratio)
    index ← list_ratio.index(max_ratio)

{ Jika ratio dari diamond yang dipilih adalah -1 (diamond merah), maka abaikan
  diamond tersebut }
if (list_ratio[index] = -1) then:
    → list_coordinates[index], True

{ Jika jumlah diamond yang dibawa adalah 4 dan diamond yang dipilih memiliki 2
  poin, maka pilih diamond sebelumnya (indeks dikurangi 1) }
if (inventory_diamonds = 4) then:
    if (list_point[index] = 2) then:
        index ← index - 1

{ Mengembalikan koordinat dari diamond yang dipilih dan status False (tidak
  mengabaikan diamond) }
→ list_coordinates[index], False
```

### c. Implementasi method red\_button

Function red\_button(position: red\_pos, position: curr\_pos, List[float]: diamond\_ratio, int: n\_diamond\_left) → boolean  
{ IS: posisi red button, rasio diamond, jumlah diamond tersisa dalam board terdefinisi }  
{ FS: akan menghasilkan true jika poin (perhitungan modifikasi sendiri untuk mencari poin) lebih dari nilai tertinggi dari rasio diamond, dan menghasilkan false untuk kondisi sebaliknya }

#### KAMUS

distance: int  
point : float

#### ALGORITMA

```
// Menghitung jarak antara posisi red button dan posisi saat ini
distance ← ((red_pos.x - curr_pos.x)^2 + (red_pos.y - curr_pos.y)^2)

// Menghitung poin berdasarkan jarak dan jumlah diamond tersisa
poin ← (2 / (distance * n_diamond_left))

// Memeriksa apakah poin lebih besar dari nilai tertinggi dalam rasio diamond
if (poin > max(diamond_ratio)) then:
    → True
else:
    → False
```

#### d. Implementasi method teleport\_use

Function teleport\_use(List[position]: diamond\_pos, List[float]: diamond\_ratio, position: teleport2\_pos, List[int]: points) → boolean  
{ IS: posisi diamond, rasio diamond, posisi teleport sebarang (teleport terjauh dari posisi bot), points dari setiap diamond terdefinisi.  
Tujuan bot untuk cari diamond, lalu bertemu teleport terdekat dalam radius yang sudah ditentukan }  
{ FS: proses akan mengecek diamond terdekat ada di posisi saat ini atau di teleport terjauh, dengan cara membandingkan nilai rasionya. }  
{ Fungsi ini akan menghasilkan true jika ratio teleport lebih besar dari ratio diamond dari current position }

#### KAMUS

ratio\_teleport: List[float]

#### ALGORITMA

```
// Menghitung rasio untuk teleport
_, ratio_teleport ← coordinate_diamond_ratio(diamond_pos, points, teleport2_pos)

// Memeriksa apakah nilai rasio maksimum untuk teleport lebih besar dari nilai rasio
// maksimum untuk diamond
if (max(ratio_teleport) > max(diamond_ratio)) then:
    → True
else:
    → False
```

#### e. Implementasi method teleport\_use\_base

Function teleport\_use\_base(position: curr\_pos, position: tel2\_pos, position: base) → boolean  
{ IS: posisi saat ini, posisi teleport sebrang (teleport terjauh dari posisi bot), posisi base. }  
{ Tujuan bot ke base, lalu bertemu teleport terdekat dalam radius yang sudah ditentukan }  
{ FS: proses akan mengecek jarak terdekat untuk pulang ke base ini memerlukan teleport atau tidak. }  
{ Fungsi ini akan menghasilkan true jika jarak terdekat ditempuh menggunakan teleport, false jika sebaliknya }

#### KAMUS

distance\_to\_base, distance\_to\_tel2: int

#### ALGORITMA

```
// Menghitung jarak dari posisi saat ini ke base dan ke teleport terjauh
distance_to_base ← (base.x - curr_pos.x)^2 + (base.y - curr_pos.y)^2
distance_to_tel2 ← (base.x - tel2_pos.x)^2 + (base.y - tel2_pos.y)^2

// Memeriksa apakah jarak ke base lebih pendek daripada jarak ke teleport terjauh
if (distance_to_base < distance_to_tel2) then:
    → True
else:
    → False
```

#### f. Implementasi method avoid\_teleport

Function avoid\_teleport(position: goal\_position, position: current\_position, int: delta\_x) → int, int  
{ IS: posisi dari goal, posisi saat ini, gerakan horizontal terdefinisi. saat ingin menuju ke tujuan lalu bertemu teleport, tapi tidak ingin melewatinya }  
{ FS: Akan menghasilkan delta\_x dan delta\_y baru untuk menghindari teleport }

#### KAMUS

selisih\_x, selisih\_y : int

#### ALGORITMA

// Jika gerakan vertikal

if (delta\_x = 0) then:

    selisih\_x ← goal\_position.x - current\_position.x

if (selisih\_x >= 0 and current\_position.x != 14) then:

        → 1, 0

else:

        → -1, 0

else:

        // Artinya delta\_x bergerak ke kanan atau ke kiri, yang mana nilai delta\_y pasti 0

        // Berarti bisa digeser atas/bawah, tergantung posisi relatif goal terhadap tujuan

        selisih\_y ← goal\_position.y - current\_position.y

if selisih\_y >= 0 and current\_position.y != 14) then:

            → 0, 1

else:

            → 0, -1

### g. Implementasi method `get_dir`

Function `get_dir(current_position, dest) → int, int`  
{ IS: posisi saat ini dan posisi tujuan terdefinisi }  
{ FS: akan menghasilkan langkah yang dilakukan untuk mencapai tujuan.  
fungsi ini adalah fungsi buatan untuk memudahkan kontrol ketika menghindari  
teleport}

#### KAMUS

`gap_x, gap_y`: int

#### ALGORITMA

```
// Menghitung perbedaan antara posisi tujuan dan posisi saat ini
gap_x ← dest.x - current_position.x
gap_y ← dest.y - current_position.y

// Memeriksa apakah pergerakan lebih dominan pada sumbu x atau sumbu y
if abs(gap_x) >= abs(gap_y) then
    // Jika pergerakan lebih dominan pada sumbu x
    → (1 if gap_x >= 0 else -1, 0)
else
    // Jika pergerakan lebih dominan pada sumbu y
    → (0, 1 if gap_y >= 0 else -1)
```

## h. Implementasi kelas BotGacor

Class BotGacor is a subclass of BaseLogic:

Constructor BotGacor():

Set directions attribute to [(1, 0), (0, 1), (-1, 0), (0, -1)]

Set goal\_position attribute to None

Set current\_direction attribute to 0

Set avoid attribute to False

Set teleport attribute to False

Function next\_move(board\_bot, board):

Set current\_position to board\_bot's position

{ Tackle }

If board\_bot has at least one diamond:

Find all enemy bots within range 2 of current\_position

Else:

Find enemy bots with at least 2 diamonds within range 2 of current\_position

If enemy\_location is not empty:

Calculate direction towards the nearest enemy bot

Return the calculated direction

{ Provide all dependencies }

Set props as board\_bot's properties

Find positions of red buttons and teleport locations

Find positions and points of diamonds in the game

Calculate diamond ratios and determine the goal position

{ Red button }

If red button should be pressed based on diamond ratios:

Set the goal position to the position of the red button

{ Back to base }

If the bot has 5 or more diamonds:

Set the goal position to the base

Else:

If the goal position is the base:

Set the goal position to the base

Else:

Set the goal position to the calculated goal position

```

{ pick algorithm after avoiding teleporter }
If avoidance flag is set:
    Calculate new delta_x and delta_y using zigzag algorithm
    Reset the avoidance flag
Else:
    Calculate new delta_x and delta_y towards the goal position using initial
    algorithm

{ policy to choose a teleporter or not }
If the teleport flag is not set and the bot has fewer than 4 diamonds:
    Iterate through teleport locations:
        If a teleport location is within range 2 of the current position:
            If the goal is not returning to base:
                Determine whether to use teleport based on diamond ratios
                Update movement direction if teleport is to be used
                Set the teleport flag to True
            Else:
                Determine whether to use teleport to return to base
                Update movement direction if teleport is to be used
                Set the teleport flag to True
        Break the loop
    Else:
        Reset the teleport flag

Return the calculated delta_x and delta_y

```



## B. Struktur Data Bot

Dalam keberjalanan game, seluruh objek disusun dalam sebuah class bernama *GameObject* yang disimpan dalam bentuk data list. Keseluruhan dari objek pada game akan dikendalikan/diatur oleh kelas bernama *Board*. Berikut adalah seluruh kelas-kelas yang digunakan dalam algoritma bot yang telah dirancang:

### 1. *GameObject*

Kelas ini berisi format kelas data untuk setiap objek di game dengan atribut-atribut berupa id (integer yang menjadi identifikasi unik untuk setiap objek), *position* (kelas/tipe data bentukan untuk koordinat objek pada game), *type* (string yang menjadi label jenis objek pada game), dan *properties* (kelas/tipe data bentukan yang berisi properti2 dari jenis2 objek tertentu pada game).

### 2. *Position*

Kelas ini merepresentasikan koordinat/posisi dari setiap objek yang ada di dalam game. Kelas ini memiliki dua atribut integer, yaitu x dan y.

### 3. *Properties*

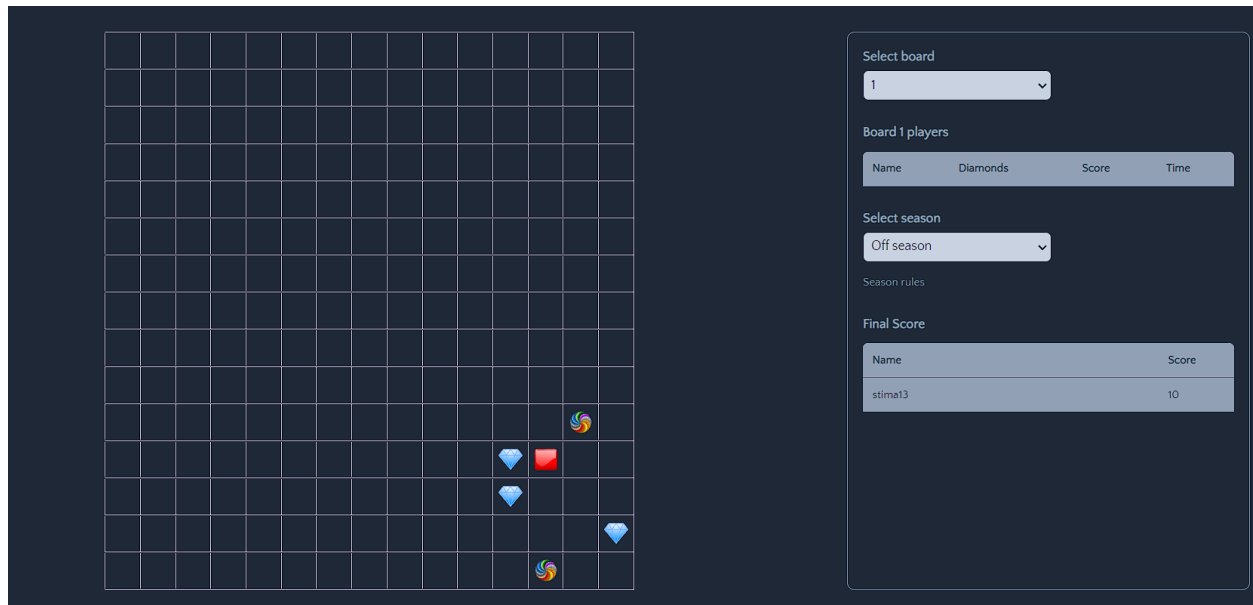
Kelas ini berisi semua properti yang relevan pada setiap objek. Kelas memiliki atribut berupa, *points* (integer yang merepresentasikan harga pada type objek diamond), *pair\_id* (integer yang merepresentasikan id pasangan dari objek portal), *score*, *inventory\_size*, *can\_tackle*, *miliseconds\_left*, *time\_joined*, dan *base*. Pada implementasi algoritma ini dan dari kelas properti, kami hanya menggunakan properti *points* dan *diamond*.

### 4. *Board*

Kelas ini merupakan kelas pengendali dari objek-objek yang berada di game. Pada keseluruhan kelas, hanya kelas *Board* yang memiliki method berupa, *get\_bot* dan *is\_valid\_move*. Method *is\_valid\_move* berguna untuk *restriction* pada objek bot agar tidak bergerak keluar dari batasan board. *Board* juga memiliki atribut berupa, *id*, *width*, *height*, *features*, *minimum\_delay\_between\_moves*, dan *game\_objects*.

## C. Hasil Pengujian dan Analisis

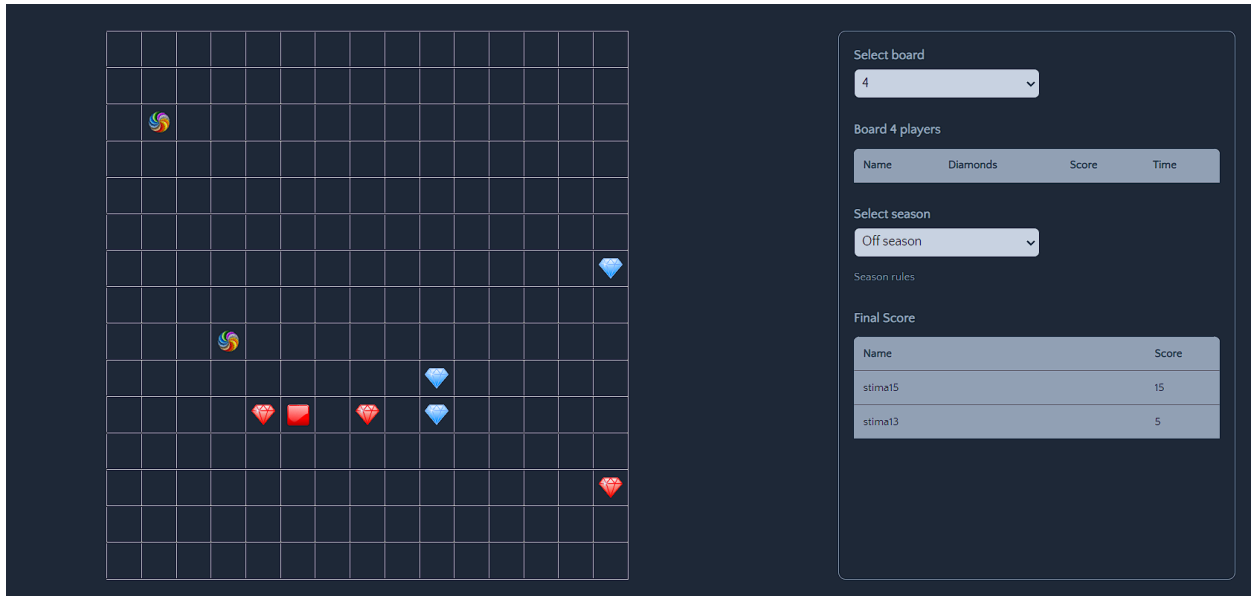
### a. Pengujian solo bot dengan algoritma utama



Pada uji pertama kita akan menguji bot dengan menggunakan algoritma utama yang sudah kami implementasikan. Pada tes pertama, bot sebetulnya sudah mendapatkan 5 diamond di inventori dan sedang menuju base. Akan tetapi waktu sudah terlanjur habis, alhasil mendapatkan skor akhir 10. Jika diamati, bot menjalankan skema sesuai harapan. Ketika bertemu teleporter maka akan mengecek apakah efisien atau tidak jika menggunakannya. Lalu jika sekiranya bot tersisa sedikit, bot juga akan mencari red button terdekat.

Terlepas dari sudah terimplementasi semua harapan kita. Permainan ini sangat mengandalkan keberuntungan, apalagi dengan waktu yang sangat singkat. Jika diamond tersebar jauh dari posisi base, maka waktu akan terkuras hanya untuk proses menuju tujuan. Alhasil banyak atau tidak skor sangat dipengaruhi dengan letak persebaran diamond

### b. Pengujian dua bot dengan algoritma yang sama



Select board  
4

Board 4 players

Name	Diamonds	Score	Time
stima15		15	
stima13		5	

Select season  
Off season

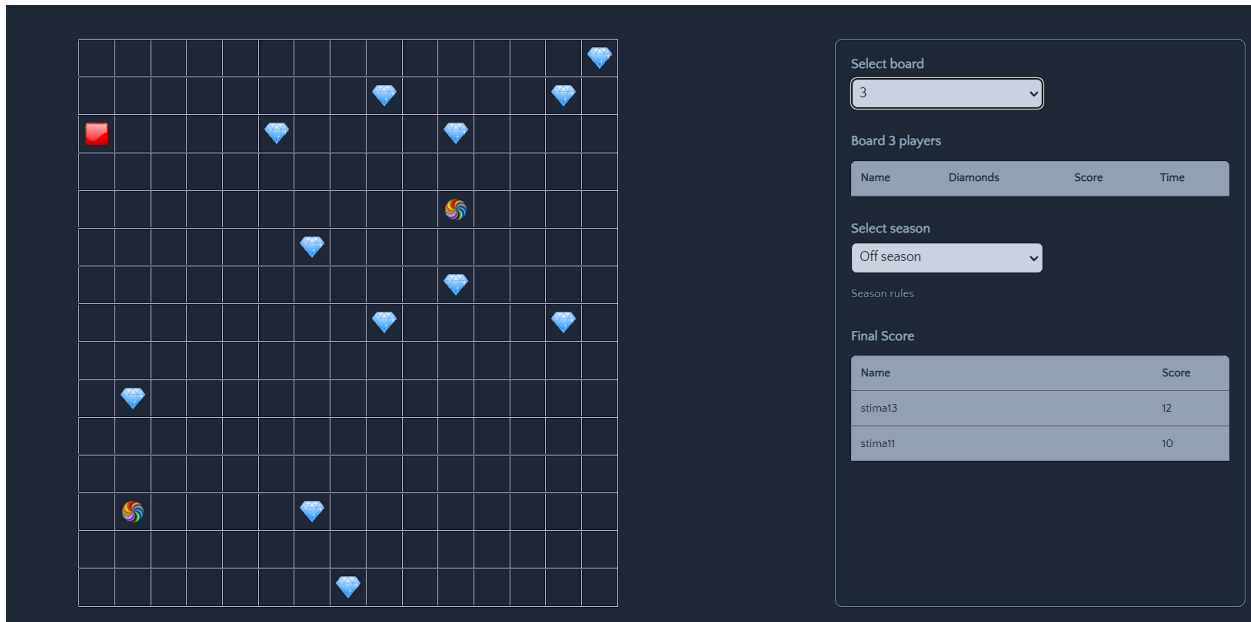
Season rules

Final Score

Name	Score
stima15	15
stima13	5

Pada tes kedua kita menggunakan implementasi dari algoritma yang sama pada kedua bot. Jika diamati terdapat selisih 10 di skor akhir. Ketimpangan yang cukup jauh ini terjadi karena terjadi *tackle* oleh stima15 kepada stima13. Karena keduanya berambisi ketika bertemu dengan bot musuh maka sempat terjadi saling kejar dan salah satunya menjadi korban *tackle*.

### c. Pengujian dua bot dengan algoritma yang berbeda



Select board  
3

Board 3 players

Name	Diamonds	Score	Time
stima13		12	
stima11		10	

Select season  
Off season

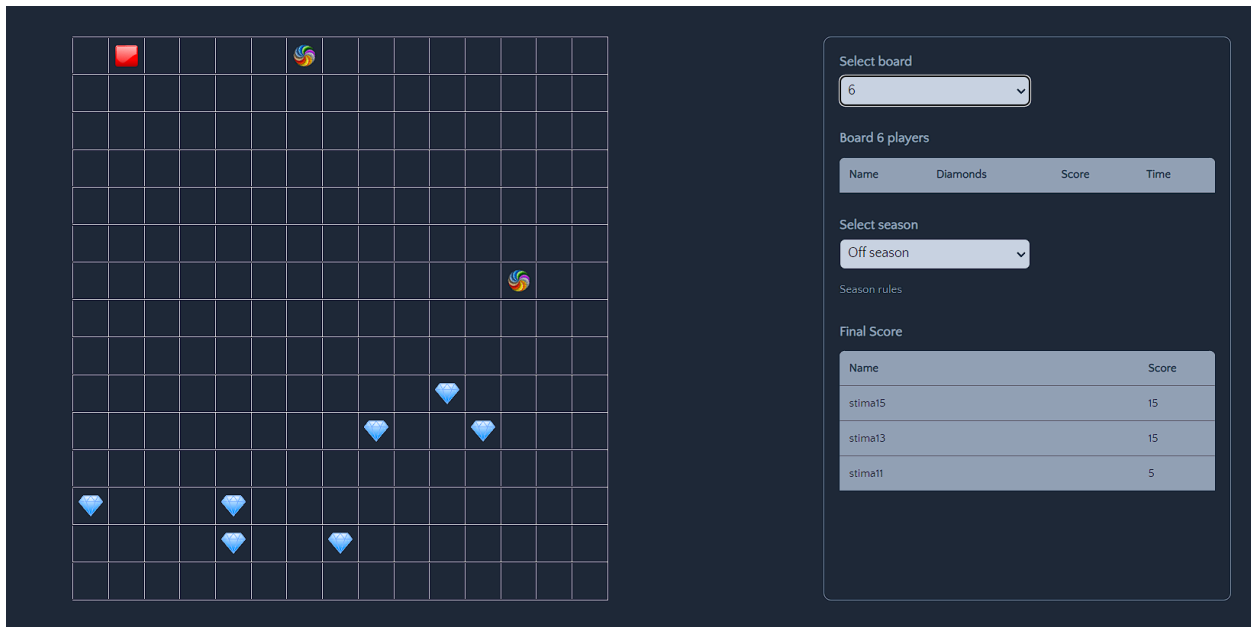
Season rules

Final Score

Name	Score
stima13	12
stima11	10

Pada tes yang ketiga saya mengimplementasikan algoritma yang berbeda dari kedua bot. Stima13 menggunakan algoritma utama (mencakup pencarian diamond dengan rasio poin per jarak paling besar, konsiderasi red button, teleporter, dan juga algoritma tackle). Sementara bot stima11 menggunakan algoritma uji (mencakup pencarian diamond dengan rasio poin per jarak paling besar, mengabaikan red button, mengabaikan teleporter, dan juga tidak ada algoritma tackle). Hasilnya dimenangkan oleh stima13 karena keefisienan jalan yang ditempuh.

#### d. Pengujian empat bot dengan algoritma campuran



The screenshot shows a game interface with a 10x10 grid. The grid contains several icons: a red button at (1,1), a teleporter at (5,1), a diamond at (1,9), a diamond at (4,9), a diamond at (5,9), a diamond at (6,8), a diamond at (7,7), a diamond at (8,6), a diamond at (9,5), a diamond at (10,4), a diamond at (10,3), a diamond at (10,2), a diamond at (10,1), a diamond at (9,1), a diamond at (8,1), a diamond at (7,1), a diamond at (6,1), a diamond at (5,1), a diamond at (4,1), a diamond at (3,1), a diamond at (2,1), a diamond at (1,1), a diamond at (1,2), a diamond at (1,3), a diamond at (1,4), a diamond at (1,5), a diamond at (1,6), a diamond at (1,7), a diamond at (1,8), a diamond at (1,9), a diamond at (1,10), a diamond at (2,2), a diamond at (2,3), a diamond at (2,4), a diamond at (2,5), a diamond at (2,6), a diamond at (2,7), a diamond at (2,8), a diamond at (2,9), a diamond at (2,10), a diamond at (3,2), a diamond at (3,3), a diamond at (3,4), a diamond at (3,5), a diamond at (3,6), a diamond at (3,7), a diamond at (3,8), a diamond at (3,9), a diamond at (3,10), a diamond at (4,2), a diamond at (4,3), a diamond at (4,4), a diamond at (4,5), a diamond at (4,6), a diamond at (4,7), a diamond at (4,8), a diamond at (4,9), a diamond at (4,10), a diamond at (5,2), a diamond at (5,3), a diamond at (5,4), a diamond at (5,5), a diamond at (5,6), a diamond at (5,7), a diamond at (5,8), a diamond at (5,9), a diamond at (5,10), a diamond at (6,2), a diamond at (6,3), a diamond at (6,4), a diamond at (6,5), a diamond at (6,6), a diamond at (6,7), a diamond at (6,8), a diamond at (6,9), a diamond at (6,10), a diamond at (7,2), a diamond at (7,3), a diamond at (7,4), a diamond at (7,5), a diamond at (7,6), a diamond at (7,7), a diamond at (7,8), a diamond at (7,9), a diamond at (7,10), a diamond at (8,2), a diamond at (8,3), a diamond at (8,4), a diamond at (8,5), a diamond at (8,6), a diamond at (8,7), a diamond at (8,8), a diamond at (8,9), a diamond at (8,10), a diamond at (9,2), a diamond at (9,3), a diamond at (9,4), a diamond at (9,5), a diamond at (9,6), a diamond at (9,7), a diamond at (9,8), a diamond at (9,9), a diamond at (9,10), a diamond at (10,2), a diamond at (10,3), a diamond at (10,4), a diamond at (10,5), a diamond at (10,6), a diamond at (10,7), a diamond at (10,8), a diamond at (10,9), a diamond at (10,10). The sidebar on the right contains the following information:

Select board: 6

Board 6 players

Name	Diamonds	Score	Time
stima15		15	
stima13		15	
stima11		5	

Select season: Off season

Season rules

Final Score

Name	Score
stima15	15
stima13	15
stima11	5

Pada tes yang terakhir saya mencoba mengimplementasikan 4 bot, stima15 dan stima13 menggunakan algoritma utama, sementara stima11 menggunakan algoritma uji, lalu sebetulnya ada bot random yang menggunakan algoritma default pemberian asisten, bot random mendapatkan 5 poin juga (Akan tetapi, tidak diketahui mengapa tidak muncul di scoreboard). Pada pengujian kali ini terlihat algoritma utama lebih dominan dari algoritma lainnya. Sempat terjadi tackle juga sehingga perolehan diamond menjadi jauh lebih besar.

## **BAB V**

### **KESIMPULAN DAN SARAN**

- **Kesimpulan**

Algoritma Greedy adalah salah satu algoritma yang kerap digunakan dalam menyelesaikan sebuah persoalan komputasional dengan tujuan untuk mengoptimasi solusi pada persoalan. Pada tugas besar ini, algoritma Greedy berhasil melakukan optimasi dalam mencari diamond terbanyak. Meskipun jalur yang diambil belum tentu jalur terbaik global, namun performa bot bisa dibilang baik dengan pertimbangan efisiensi eksekusi

- **Saran**

Terdapat beberapa saran dalam pengerjaan tugas besar selanjutnya :

- Pembuatan laporan bisa dicicil mengingat penilaian dalam laporan terkait reasoning dari algoritma greedy yang dibuat cukup besar
- Lebih banyak melakukan pengujian dengan bot kelompok lain

## LAMPIRAN

- Tautan repository Github : [https://github.com/Akmal2205/Tubes1\\_sambut-ramadhan](https://github.com/Akmal2205/Tubes1_sambut-ramadhan)
- Tautan video : <https://youtu.be/ikU8PiUMh8M>

## DAFTAR PUSTAKA

Levitin, Anany. *Introduction to the Design & Analysis of Algorithms*. Addison-Wesley, 2003.

Munir, Rinaldi. *Algoritma Greedy 2021*. Diakses pada 1 maret 2024 pada

[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)