

PEMANFAATAN ALGORITMA *GREEDY* DALAM PEMBUATAN BOT PERMAINAN ‘DIAMONDS’



Oleh:
Kelompok (**sem-4-dapat-silverqueen.semoga**)

13522013 Denise Felicia Tiowanni

13522063 Shazya Audrea Taufik

13522087 Shulha

Dosen Pengampu : Dr. Ir. Rinaldi Munir, M.T
IF2211 - Strategi Algoritma

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

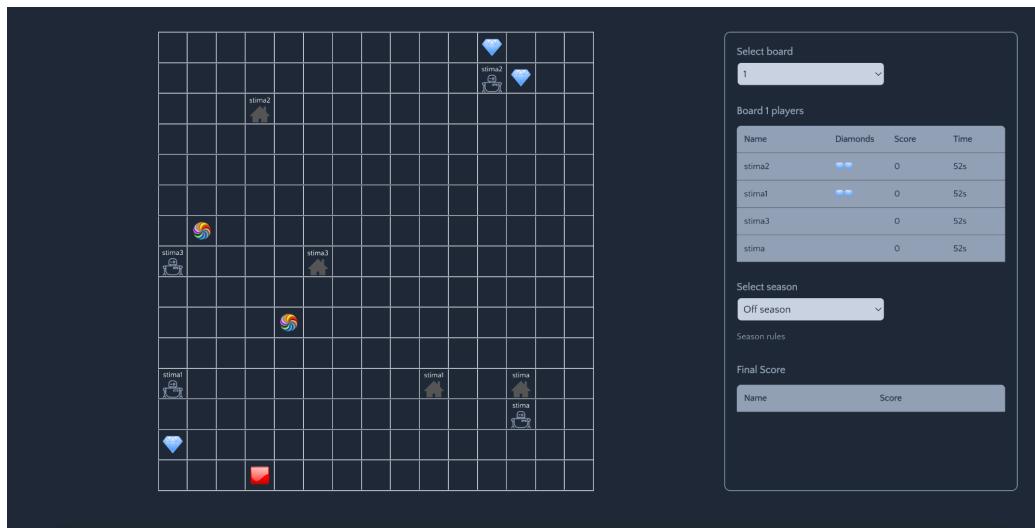
DAFTAR ISI

DAFTAR ISI	2
BAB 1	
DESKRIPSI MASALAH	3
BAB 2	
TEORI DASAR	4
2.1. Algoritma Greedy	4
2.2. Game Engine	4
2.3. Permainan Diamonds	5
2.4. Cara Kerja Bot dalam Permainan Diamonds dengan Algoritma Greedy	6
BAB 3	
APLIKASI STRATEGI GREEDY	7
3.1. Mapping Persoalan Diamonds ke Elemen Algoritma Greedy	7
3.2. Eksplorasi Alternatif Solusi Greedy	7
3.3. Analisis Efisiensi dan Efektivitas Alternatif Solusi	15
3.4. Pemilihan Strategi Greedy	22
BAB 4	
IMPLEMENTASI DAN PENGUJIAN	27
4.1. Implementasi dalam Pseudocode	27
4.2. Struktur Data Program	30
4.3. Analisis dan Pengujian	33
BAB 5	
PENUTUP	37
5.1. Kesimpulan	37
5.2. Saran	37
5.3. Refleksi	37
DAFTAR PUSTAKA	38
LAMPIRAN	39

BAB 1

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 sederhana, melainkan 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.



Gambar 1. Permainan Diamonds

Program permainan Diamonds ini sendiri terdiri atas *game engine*, yang secara umum berisi kode backend permainan (logic serta API), kode frontend permainan (visual permainan), serta *bot starter pack* yang berisi program untuk memanggil API pada backend, bot logic, maupun program utama (main) dan utilitas lainnya.

Pada Tugas Besar 1 Strategi Algoritma ini, kami diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan dengan menggunakan Algoritma Greedy. Dalam mengimplementasikan bot, kami diminta untuk menggunakan bahasa pemrograman Python. Template bot sendiri sudah tersedia pada *game engine* sehingga kami hanya mengubah strategi Greedy bot dalam memenangkan permainan.

BAB 2

TEORI DASAR

2.1.Algoritma Greedy

Algoritma *greedy* adalah algoritma yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Algoritma greedy memecahkan persoalan optimasi secara langkah per langkah (step by step) sedemikian sehingga, pada setiap langkah, diambil pilihan yang terbaik pada saat itu (optimum lokal) tanpa memperhatikan konsekuensi ke depan dengan harapan bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Elemen - elemen pada algoritma greedy :

1. Himpunan Kandidat (C) : berisi kandidat yang akan dipilih pada setiap langkah algoritma.
2. Himpunan Solusi : 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 dan strategi bersifat heuristik.
5. Fungsi Kelayakan (*feasible*) : Memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi Obyektif : mencari solusi yang paling efektif.

Contoh - contoh persoalan yang diselesaikan dengan algoritma greedy antara lain adalah persoalaan penukaran uang, persoalan memilih aktivitas, minimasi waktu di dalam sistem, persoalan knapsack (knapsack problem), penjadwalan Job dengan tenggat waktu, pohon merentang minimum, lintasan terpendek, kode Huffman, dan pecahan mesir.

2.2.Game Engine

Game Engine adalah sistem perangkat lunak yang dirancang untuk menciptakan dan mengembangkan sebuah game. *Game Engine* pada dasarnya adalah sebuah *library* yang dapat digunakan untuk membuat permainan. *Game engine* memberikan kemudahan bagi *game developer* karena menyediakan fungsi-fungsi inti dari sebuah permainan, misalnya

grafika (menghasilkan grafika 2-dimensi atau 3-dimensi), fisika (menghitung dan menyimulasikan hukum-hukum gerak dan hukum fisika lainnya), audio, atau kecerdasan buatan. *Game Engine* dapat digunakan untuk membuat lebih dari satu permainan, dan pengembang permainan dapat mengoptimalkan proses pengembangan dengan cara menggunakan atau mengadaptasi *game engine* yang telah ada sebelumnya.

2.3. Permainan Diamonds

Permainan Diamonds berisi bot-bot yang memiliki objektif untuk memperoleh sebanyak-banyaknya diamonds. Permainan Diamonds memiliki komponen-komponen sebagai berikut:

- 1. Diamonds**

Untuk memenangkan pertandingan, bot harus mengumpulkan diamond 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. Bots and Bases**

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.

- 3. 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.

- 4. Teleporter**

Pada board, terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

- 5. Inventory**

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

Cara kerja permainan Diamonds adalah sebagai berikut:

1. Setiap 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 memiliki waktu bergerak yang sama (misal 60 detik).
3. Setelah mengumpulkan bot, apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
4. Jika suatu bot menimpa posisi bot lain, bot lain tersebut akan dikirim ke home base dan semua diamond pada inventory bot lain tersebut akan hilang, diambil masuk ke inventory bot atau *tackle*.
5. Permainan berakhir saat waktu seluruh bot habis. Skor masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

2.4.Cara Kerja Bot dalam Permainan Diamonds dengan Algoritma Greedy

Untuk memenangkan permainan, bot akan berjalan menurut langkah tertentu untuk mencapai dan memperoleh sebanyak-banyaknya diamond dan membawanya ke home base dengan aman. Pada permainan ini, bot didesain agar bisa mencapai diamonds dengan algoritma *greedy*. Hal tersebut berarti bot akan mengambil langkah menuju diamond yang dirasa paling optimal untuk diraih tanpa memperhatikan langkah ke depan apakah paling optimal atau bukan (*take what is most optimal now*). Berdasarkan algoritma *greedy*, bot akan mencari titik tujuan yang dirasa paling optimal (baik itu diamond, home base, maupun objek-objek lainnya). Bot kemudian akan mengambil langkah menuju titik tujuan tersebut baik dengan melangkah ke kanan, kiri, atas, ataupun bawah. Berdasarkan algoritma tertentu, bot bisa juga mengambil langkah untuk melakukan *tackle* terhadap bot lawan maupun menghindari bot lawan.

BAB 3

APLIKASI STRATEGI *GREEDY*

3.1. Mapping Persoalan Diamonds ke Elemen Algoritma Greedy

Himpunan Kandidat	Semua koordinat yang menuju diamonds atau base.
Himpunan Solusi	Semua koordinat menuju diamonds atau base yang sudah dipilih (mengutamakan diamonds merah daripada diamonds biru)
Fungsi Solusi	Memeriksa apakah diamond sudah berhasil diambil
Fungsi Seleksi (<i>selection function</i>)	Melakukan seleksi dari semua koordinat yang menuju diamonds berdasarkan strategi <i>greedy</i> yang dipakai.
Fungsi Kelayakan (<i>feasible</i>)	Memeriksa apakah masih dapat mengumpulkan diamond (<i>inventory</i> belum penuh)
Fungsi Obyektif	Mengambil sebanyak-banyaknya diamonds dan menyimpannya di base

3.2. Eksplorasi Alternatif Solusi Greedy

1. Alternatif 1

a. Penjelasan Umum Strategi

Salah satu alternatif algoritma yang dapat digunakan adalah algoritma *greedy by distance*. Pendekatan algoritma ini mengutamakan jarak terdekat antara bot dengan *diamonds* apapun dengan bot terlepas dari tipe *diamonds* tersebut.. Program akan menghitung jarak antara bot dengan seluruh *diamonds* dan memilih yang terdekat lalu menentukan langkah yang tepat

menuju diamonds yang terdekat tersebut. Jika bot berhasil mengambil diamond, bot akan mencari kembali diamonds lain dengan jarak terdekat.

b. Mapping Elemen Algoritma Greedy

Fungsi Seleksi (<i>selection function</i>)	Menghitung jarak bot ke koordinat seluruh diamonds, melakukan seleksi diamonds terdekat dan menentukan <i>step</i> koordinat menuju diamonds tersebut
Fungsi Obyektif	Mengambil sebanyak-banyaknya diamonds dan menyimpannya di base

c. Implementasi Algoritma

Program menyimpan data koordinat seluruh diamonds yang terdapat pada board dalam sebuah list. Program kemudian mendapatkan diamonds dengan jarak minimum dari list tersebut dan menyimpan koordinat tujuan. Program kemudian memilih langkah untuk menuju koordinat tujuan tersebut. Terdapat beberapa modifikasi agar langkah yang dipilih tidak menyebabkan *invalid move* yaitu bergerak diagonal ataupun menabrak batas papan.

```
def next_move(self, board_bot: GameObject, board: Board):
    props = board_bot.properties
    # Analyze new state
    if props.diamonds == 5:
        # Move to base
        base = board_bot.properties.base
        self.goal_position = base
    else:
        # Go to nearest diamond
        list_of_diamonds = board.diamonds
        diamonds_distance = []
        current_position = board_bot.position
        for d in list_of_diamonds:
            diamonds_distance.append(abs(current_position.x - d.position.x) + abs(current_position.y - d.position.y))
        nearest_idx = diamonds_distance.index(min(diamonds_distance))
        self.goal_position = list_of_diamonds[nearest_idx].position
        current_position = board_bot.position
```

```

# We are aiming for a specific position, calculate delta
delta_x, delta_y = get_direction(
    current_position.x,
    current_position.y,
    self.goal_position.x,
    self.goal_position.y,
)

#Invalid Move Handler
if (0 < board_bot.position.x + delta_x < board.width) and (0
< board_bot.position.y + delta_y < board.height):
    return delta_x, delta_y
else:
    # Adjust movement to stay within board boundaries
    while (delta_x + board_bot.position.x >= board.width) or
(delta_x + board_bot.position.x < 0) or (delta_y + board_bot.position.y
>= board.height) or (delta_y + board_bot.position.y < 0):
        delta_x = random.randint(-1, 1)
    if delta_x != 0:
        delta_y = 0
    else:
        delta_y = random.choice([-1, 1])
return delta_x, delta_y

```

2. Alternatif 2

a. Penjelasan Umum Strategi

Salah satu alternatif algoritma yang dapat digunakan adalah algoritma *greedy by diamonds*. Algoritma ini akan membuat bot pemain bergerak mencari diamonds yang jaraknya paling dekat dengan mendahulukan diamonds merah daripada diamonds biru.

b. Mapping Elemen Algoritma *Greedy*

Fungsi Seleksi (<i>selection function</i>)	Menghitung jarak bot ke koordinat seluruh diamonds, melakukan seleksi diamonds terdekat dan mendahulukan diamond berwarna merah dan menentukan <i>step</i> koordinat menuju diamonds tersebut
Fungsi Obyektif	Mengambil sebanyak-banyaknya diamonds dan menyimpannya di base

Fungsi Seleksi (<i>selection function</i>)	Menghitung jarak bot ke koordinat seluruh diamonds, melakukan seleksi diamonds terdekat dan mendahulukan diamond berwarna merah dan menentukan step koordinat menuju diamonds tersebut
	dengan mengutamakan diamonds merah

c. Implementasi Algoritma

Algoritma yang digunakan sama dengan alternatif *greedy by distance* dengan memilih diamond merah terdekat sebagai koordinat tujuan. Ketika menemukan diamond merah, diamond merah terdekat akan didahulukan dari diamond biru dengan jarak yang sama. Algoritma ini juga mempertimbangkan inventory yang sudah terisi 4 diamonds untuk tidak mengambil diamond merah karena tidak akan muat di inventory. Implementasi algoritma sama dengan alternatif *by distance* dengan fungsi mencari diamond merah atau biru terdekat sebagai berikut.

```
def find_nearest_diamond(self, board_bot: GameObject, board: Board):
    diamonds = board.diamonds
    if diamonds:
        if board_bot.properties.diamonds == 4:
            # kalau diamond count sudah 4, ambil yang biru aja
            gausah yang merah lagi
            blue_diamonds = [diamond for diamond in diamonds if
diamond.properties.points != 2]
            if blue_diamonds:
                nearest_diamond = min(blue_diamonds, key=lambda
diamond: calculate_distance(diamond.position, board_bot.position))
            else:
                nearest_diamond = min(diamonds, key=lambda
diamond: calculate_distance(diamond.position, board_bot.position))
            else:
                nearest_diamond = min(diamonds, key=lambda diamond:
calculate_distance(diamond.position, board_bot.position))
                self.goal_position = nearest_diamond.position
            else:
                # No diamonds found, roam around
                self.goal_position = None
```

3. Alternatif 3

a. Penjelasan Umum Strategi

Alternatif algoritma lain yang dapat digunakan adalah algoritma *greedy by defence*. Algoritma ini modifikasi tambahan dari alternatif algoritma yang sebelumnya sudah dijelaskan. Program akan memetakan seluruh jarak bot lain di papan dan mendeteksi apabila ada bot pemain lain yang berdekatan dengan bot tersebut. Jika ditemukan bot pemain lain yang berdekatan, bot akan mengambil langkah menjauh. Kemudian, jika bot sudah memiliki 3 atau lebih diamonds dan jarak dia ke base lebih dekat daripada jarak ke diamond terdekat, maka bot akan mengutamakan untuk kembali ke base untuk menjaga diamond yang telah dimilikinya.

b. Mapping Elemen Algoritma *Greedy*

Fungsi Seleksi (<i>selection function</i>)	Memetakan seluruh koordinat bot dan mengecek apa ada bot yang berdekatan. Memilih langkah menjauh dari bot pemain lain jika terdapat bot berdekatan. Melakukan aksi seperti algoritma <i>greedy by diamonds</i>
Fungsi Obyektif	Menjauh dari bot lain sembari mengambil sebanyak-banyaknya diamonds dan menyimpannya di base dengan mengutamakan diamonds merah

c. Implementasi Algoritma

Algoritma yang digunakan sama dengan alternatif *greedy by defence* dengan mengecek terlebih dahulu jika ada bot pemain lain yang berjarak satu langkah dari bot. Jika terdapat, bot akan bergerak menjauh dari bot pemain lain tersebut dengan tujuan menjaga diamonds yang sedang dibawanya.

```
def next_move(self, board_bot: GameObject, board: Board):
    props = board_bot.properties
```

```

bots = board.bots
# Check other bots position
for bot in bots:
    if bot != board_bot:
        temp_distance = calculate_distance(bot.position,
board_bot.position)
        if temp_distance <= 1:
            temp_delta_x = bot.position.x -
board_bot.position.x
            temp_delta_y = bot.position.y -
board_bot.position.y
            # If adjacent, move away
            if temp_delta_x == 0 and abs(temp_delta_y) == 1:
                # jika ada bot lawan di sebelah kanan atau
kiri, arahkan untuk bergerak ke atas atau bawah
                if 0 <= board_bot.position.y - temp_delta_y
<= board.width:
                    return 0, -temp_delta_y
                elif 0 <= board_bot.position.y +
temp_delta_y <= board.width:
                    return 0, temp_delta_y
                elif temp_delta_y == 0 and abs(temp_delta_x) == 1:
                    # jika ada bot lawan di atas atau bawah,
arahkan untuk bergerak ke kanan atau kiri (atur invalid move juga)
                    if 0 <= board_bot.position.x - temp_delta_x
<= board.width:
                        return -temp_delta_x, 0
                    elif 0 <= board_bot.position.x +
temp_delta_x <= board.width:
                        return temp_delta_x, 0

        if props.diamonds == 5:
            # Move to base
            base = board_bot.properties.base
            self.goal_position = base
        else:
            # Find the nearest diamond
            self.find_nearest_diamond(board_bot, board)
            if calculate_distance(self.goal_position,
board_bot.position) > calculate_distance(board_bot.properties.base,
board_bot.position) and props.diamonds >= 3:
                self.goal_position = board_bot.properties.base

            current_position = board_bot.position
            if self.goal_position:
                # We are aiming for a specific position, calculate delta
delta_x, delta_y = get_direction(
                    current_position.x,
                    current_position.y,
                    self.goal_position.x,
                    self.goal_position.y,

```

```

        )
    else:
        # Roam around
        delta = self.directions[self.current_direction]
        delta_x = delta[0]
        delta_y = delta[1]
        if random.random() > 0.6:
            self.current_direction = (self.current_direction +
1) % len(
            self.directions
        )

        if (0 < board_bot.position.x + delta_x < board.width) and
(0 < board_bot.position.y + delta_y < board.width):
            return delta_x, delta_y
        else:
            # Adjust movement to stay within board boundaries
            while (delta_x + board_bot.position.x >= board.width) or
(delta_x + board_bot.position.x < 0) or (delta_y + board_bot.position.y
>= board.width) or (delta_y + board_bot.position.y < 0):
                delta_x = random.randint(-1, 1)
                if delta_x != 0:
                    delta_y = 0
                else:
                    delta_y = random.choice([-1, 1])
            return delta_x, delta_y

```

4. Alternatif 4

a. Penjelasan Umum Strategi

Alternatif algoritma lain yang dapat digunakan adalah dengan modifikasi tambahan dari alternatif algoritma yang sebelumnya sudah dijelaskan. Alternatif ini akan memanfaatkan komponen teleporter pada papan. Teleporter sendiri berfungsi memindahkan posisi bot ke teleporter yang lain.

b. Mapping Elemen Algoritma *Greedy*

Fungsi Seleksi (<i>selection function</i>)	Memetakan seluruh koordinat diamonds dan teleporter. Memilih teleporter atau diamonds sebagai koordinat tujuan (menyesuaikan jarak dan kebutuhan).
--	--

Fungsi Obyektif	Mengambil sebanyak-banyaknya diamonds dan menyimpannya di base dengan mengutamakan diamonds merah dan memanfaatkan teleporter
-----------------	---

c. Implementasi Algoritma

Algoritma tambahan yang dilakukan adalah untuk mengecek apakah teleport akan dilangkahi selama perjalanan menuju diamond. Jika diamonds tujuan sebenarnya sudah dekat dengan posisi bot saat ini, maka menginjak atau melangkahi teleporter mungkin akan mengakibatkan kerugian. Oleh karena itu, bot dicegah untuk menginjak teleporter jika diamonds yang menjadi tujuan sudah berjarak dekat dengan bot (dengan dekat dalam algoritma ini adalah kurang dari delapan langkah).

```

# Mencari posisi teleport
def find_teleport_gameobject(self, board: Board) -> Optional[Position]:
    teleport_objects = [obj for obj in board.game_objects if
        obj.type == "TeleportGameObject"]
    if teleport_objects:
        return teleport_objects[0].position,
    teleport_objects[1].position
    else:
        return None

# Mendapatkan lokasi teleport game object
teleport_position, teleport_position1 = self.find_teleport_gameobject(board)
if teleport_position:
    print("teleporter location:", teleport_position)
    teleport_x = teleport_position.x
    teleport_y = teleport_position.y
    print("teleporter location - x:", teleport_x, "y:", teleport_y)

if teleport_position1:
    print("teleporter location:", teleport_position1)
    teleport_x1 = teleport_position1.x
    teleport_y1 = teleport_position1.y
    print("teleporter location - x:", teleport_x1, "y:", teleport_y1)

```

```

#Menentukan langkah
if self.goal_position:
    # We are aiming for a specific position, calculate delta
    delta_x, delta_y = get_direction(
        current_position.x,
        current_position.y,
        self.goal_position.x,
        self.goal_position.y,
    )
    board_x, board_y = board_bot.position.x,
    board_bot.position.y
    if (board_x + delta_x == teleport_x and board_y + delta_y == teleport_y) or \
        (board_x + delta_x == teleport_x1 and board_y + delta_y == teleport_y1):
        distance = calculate_distance(self.goal_position,
            board_bot.position)
        if distance > 8 and (0 <= board_x + delta_x <= 14) and (0 <= board_y + delta_y <= 14):
            return -delta_x, -delta_y
        else:
            return delta_x, delta_y

# (Cont...) Penyesuaian invalid move seperti algoritma lainnya

```

3.3. Analisis Efisiensi dan Efektivitas Alternatif Solusi

1. Alternatif 1

a. Kompleksitas Algoritma

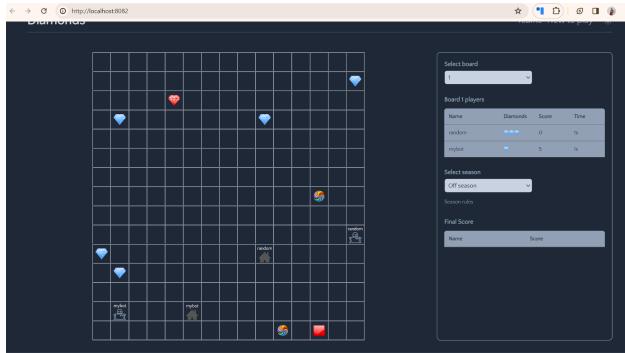
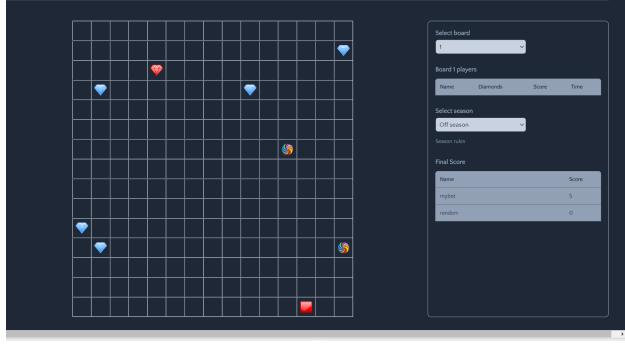
Kompleksitas waktu dari kode ini dapat dipecah sebagai berikut:

- Memeriksa apakah bot memiliki 5 diamond: O(1)
- Menemukan diamond terdekat:
 - Iterasi seluruh diamond: O(n) di mana n adalah jumlah diamond.
 - Menghitung jarak setiap diamond: O(1) per diamond, jadi totalnya O(n).
- Menghitung delta untuk pergerakan: O(1)
- Memeriksa apakah perpindahan tersebut berada dalam batas papan: O(1)
- Menyesuaikan pergerakan agar tetap berada dalam batas papan:
 - While loop akan berjalan paling banyak 4 kali (sekali untuk setiap batas), dan di dalam perulangan, terdapat dua panggilan random.randint atau random.choice, masing-masing dengan

kompleksitas waktu konstan $O(1)$. Jadi, kompleksitas waktunya adalah $O(1)$ untuk kasus terburuk.

Secara keseluruhan, kompleksitas waktu dari kode ini adalah $O(n)$ dengan n adalah jumlah diamond.

b. Eksperimen Singkat

Pengambilan Diamond	Dengan membandingkan solusi alternatif 1 dengan algoritma bawaan program, bot tidak lagi bergerak “tanpa arah”, melainkan bot dapat langsung menuju lokasi diamond yang terdekat.
Pada waktu tersisa 5 detik, bot berhasil menangkap 6 diamonds dibandingkan dengan bot random yang baru menangkan 3 diamonds	
Saat waktu habis, bot sudah mengumpulkan 5 diamonds di home base dibandingkan dengan bot random yang belum memiliki satupun diamond di home base	

c. Hasil Analisis

Algoritma ini efektif jika diamond-diamond yang terdapat pada *board* memang dekat dengan posisi bot saat ini.

2. Alternatif 2

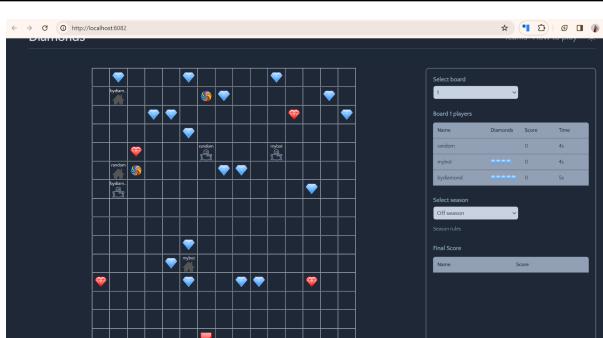
a. Kompleksitas Algoritma

Kompleksitas waktu dari kode tersebut dapat dipecah sebagai berikut:

- Menginisialisasi variabel seperti *diamonds*: O(1)
- Memeriksa apakah ada *diamond*: O(1)
- Memeriksa apakah bot memiliki 4 *diamond*: O(1)
- Memfilter diamond biru (jika bot memiliki 4 *diamond*): O(n) dimana n adalah jumlah diamond. Operasi ini mengulangi semua diamond dan menyaring diamond biru.
- Menemukan diamond terdekat: O(n) di mana n adalah jumlah diamond. Operasi ini menggunakan fungsi min dengan fungsi lambda yang menghitung jarak setiap diamond, yang membutuhkan waktu konstan O(1).

Secara keseluruhan, algoritma ini memiliki kompleksitas O(n) termasuk dengan bagian fungsi *next_move* yang menggunakan algoritma greedy by distance.

b. Eksperimen Singkat

Pengambilan Diamond	Dengan membandingkan solusi alternatif 2 dengan solusi sebelumnya, bot tidak hanya bergerak mengambil diamond yang terdekat saja, melainkan juga memprioritaskan pengambilan diamond merah apabila jarak ke diamond merah dan biru sama.
Pada waktu tersisa 4 detik, bot alternatif ini memimpin dengan memperoleh 5 diamonds dibandingkan dengan bot alternatif sebelumnya yang baru mengumpulkan 4	

diamonds	
Pada akhir permainan, tidak ada bot yang berhasil membawa diamonds ke base	

c. Hasil Analisis

Algoritma ini efektif jika terdapat diamond merah dan biru di sekitar bot yang mempunyai jarak sama, karena dengan jumlah langkah yang sama, bot dapat mengumpulkan lebih banyak poin. Algoritma alternatif ini tidak terlalu berpengaruh jika dibandingkan dengan algoritma alternatif sebelumnya. Pada permainan Diamonds, faktor keuntungan kondisi awal yang dilakukan secara otomatis melalui *game engine* juga mempengaruhi hasil permainan.

3. Alternatif 3

a. Kompleksitas Algoritma

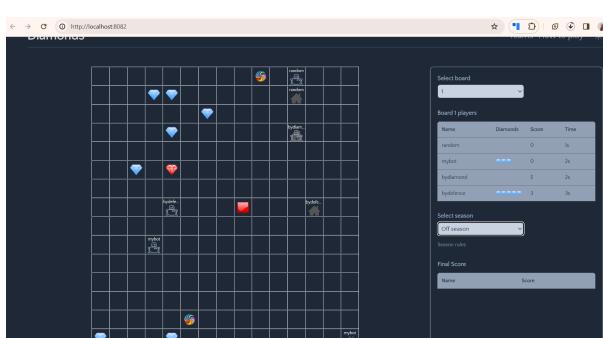
Kompleksitas waktu dari kode ini dapat dipecah sebagai berikut:

- Menginisialisasi variabel seperti props dan bots: $O(1)$
- Mengecek posisi bot lain:
 - Iterasi terhadap semua bot: $O(n)$ dengan n adalah jumlah bot.
 - Mencari jarak ke setiap bot lainnya: $O(1)$ per bot, jadi totalnya $O(n)$.
- Mengecek kondisi terkini:
 - Mengecek apakah bot memiliki 5 diamond: $O(1)$
 - Mencari diamond terdekat: $O(n)$
 - Menghitung jarak: $O(1)$
- Mengecek apakah bot tetap berada dalam batas papan: $O(1)$
- Menyesuaikan gerakan agar tetap berada di dalam batas papan:

- While loop akan berjalan paling banyak 4 kali (sekali untuk setiap batas), dan di dalam perulangan, terdapat dua panggilan random.randint atau random.choice, masing-masing dengan kompleksitas waktu konstan O(1). Jadi, kompleksitas waktunya adalah O(1) untuk kasus terburuk.

Secara keseluruhan, kompleksitas waktu dari kode ini adalah O(n) dengan n adalah jumlah diamond.

b. Eksperimen Singkat

Menjauhi Lawan	Dengan membandingkan solusi alternatif 3 dengan solusi-solusi alternatif sebelumnya, sekarang bot tidak hanya bergerak mengumpulkan diamond saja, tetapi juga mempertimbangkan posisi bot lawan. Dengan demikian, bot dapat menghindari bot lawan agar tidak tertabrak dan dapat menjaga diamond yang sudah terkumpul.
Pada waktu tersisa 2 detik, perolehan diamond adalah ByDistance = 3 ByDiamond = 5 ByDefence = 6 Random = 0	 A screenshot of a game interface titled "Diamonds". It shows a 10x10 grid with several blue diamond icons scattered across it. A red diamond icon is also present. On the right side of the screen, there is a sidebar with the following information: <ul style="list-style-type: none">"Select board": dropdown menu set to "1"."Board players": table with three rows:<ul style="list-style-type: none">robot: Diamonds 0, Score 1, Time 2shybridized: Diamonds 0, Score 2, Time 2ssymbolic: Diamonds 3, Score 3, Time 2s"Select session": dropdown menu set to "off session"."Score rules": dropdown menu set to "None"."Final Score": table with one row: Name Score.

<p>Pada eksperimen lain, hasil yang diperoleh berbeda. Eksperimen di samping menunjukkan ByDefence kalah bahkan dengan mybot (ByDistance/alternatif 1)</p>	
--	--

c. Hasil Analisis

Algoritma ini efektif jika bot pemain berada didekat bot lawan, sehingga tidak terjadi tabrakan atau mengurangi kemungkinan bot mengalami terkena *tackle* oleh lawan. Pada permainan dengan sedikit bot lawan, algoritma ini mungkin kurang berguna, namun dapat sangat mengantisipasi hilangnya diamonds pada battle yang lebih besar. Perolehan diamonds oleh bot juga sangat mungkin disebabkan oleh letak awal bot dan home base yang random.

4. Alternatif 4

a. Kompleksitas Algoritma

- Menginisialisasi variabel seperti props dan bots dan teleport: $O(1)$
- Mengecek posisi teleporter dengan iterasi terhadap dua *teleporter*: $O(n)$
- Mengecek posisi bot lain:
 - Iterasi terhadap semua bot: $O(n)$ dengan n adalah jumlah bot.
 - Mencari jarak ke setiap bot lainnya: $O(1)$ per bot, jadi totalnya $O(n)$.
- Mengecek kondisi terkini:
 - Mengecek apakah bot memiliki 5 diamond: $O(1)$
 - Mencari diamond terdekat: $O(n)$
 - Menghitung jarak: $O(1)$
- Mengecek apakah bot tetap berada dalam batas papan: $O(1)$
- Menyesuaikan gerakan agar tetap berada di dalam batas papan:

- While loop akan berjalan paling banyak 4 kali (sekali untuk setiap batas), dan di dalam perulangan, terdapat dua panggilan random.randint atau random.choice, masing-masing dengan kompleksitas waktu konstan $O(1)$. Jadi, kompleksitas waktunya adalah $O(1)$ untuk kasus terburuk.

Secara keseluruhan, kompleksitas waktu dari kode ini adalah $O(n)$ dengan n adalah jumlah diamond.

b. Eksperimen Singkat

Pengambilan Diamond	Dengan membandingkan solusi alternatif 4 dengan solusi-solusi alternatif sebelumnya, bot tidak hanya bergerak mengambil diamond yang terdekat saja, melainkan juga memprioritaskan pengambilan diamond merah apabila jarak ke diamond merah dan biru sama.
Menjauhi Lawan	Dengan membandingkan solusi alternatif 4 dengan solusi-solusi alternatif sebelumnya, sekarang bot tidak hanya bergerak mengumpulkan diamond saja, tetapi juga mempertimbangkan posisi bot lawan. Dengan demikian, bot dapat menghindari bot lawan agar tidak tertabrak dan dapat menjaga diamond yang sudah terkumpul.
Menjauhi <i>Teleporter</i>	Berbeda dengan solusi-solusi sebelumnya, pada solusi alternatif 4 ini pula, bot akan bergerak menjauhi <i>teleporter</i> apabila posisi diamond jauh dari <i>teleporter</i> .

<p>Pada eksperimen dimana bot superqueen (gabungan semuanya) melawan bot defence dan diamonds, bot superqueen menang dengan perolehan 15 diamonds.</p>	
--	--

c. Hasil Analisis

Algoritma ini efektif apabila saat posisi *teleporter* jauh dari diamond. Algoritma ini juga efektif sehingga ketika kita ingin mengejar diamond, kita tidak dipindahkan atau dijauhkan dengan adanya teleport.

3.4.Pemilihan Strategi Greedy

Dari semua alternatif strategi *greedy* yang ada dan sudah dijelaskan, dapat dibuat sebuah algoritma *greedy* yang merupakan **gabungan** dari semua algoritma yang ada agar dapat saling menutupi kekurangan atau kelemahan satu sama lain. Implementasi ini mempunyai alur dengan urutan prioritas sebagai berikut.

1. Menghitung jarak antara bot pemain dengan diamonds.
2. Mengambil jarak terdekat bot pemain dengan diamonds.
3. Membagi diamond menjadi merah dan biru. Diamond dengan jarak yang sama akan diutamakan diamond merah.
4. Saat bot pemain sudah menampung minimal 3 dari 5 diamond dan jarak ke base lebih dekat daripada ke diamond lain, program akan kembali ke base.
5. Program akan menghindar dari bot lawan.
6. Program mengatasi *invalid move* yang mungkin terjadi.
7. Program akan menjauhi *teleporter*.

Seluruh solusi pada implementasi diatas dibuat berdasarkan solusi-solusi *greedy* yang telah dijabarkan sebelumnya. Program yang menghitung jarak antara bot dengan diamonds dan mengambil jarak yang terdekat merupakan implementasi *greedy by*

distance. Apabila jarak suatu diamond merah sama dengan diamond biru, program akan membuat bot bergerak memprioritaskan diamond merah. Program juga membuat bot pemain kembali ke base apabila diamond yang sudah terkumpul lebih besar atau sama dengan 3 dan jarak menuju base lebih dekat daripada jarak menuju diamond selanjutnya agar meminimalisir kemungkinan diamond “tercuri”. Kedua hal ini merupakan implementasi dari *greedy by diamond*. Selanjutnya adalah implementasi dari *greedy by defence*, dimana program juga menghitung jarak antara bot pemain dengan bot lawan, sehingga apabila bot lawan berada tepat di sebelah bot pemain, bot pemain dapat bergerak menghindari bot lawan sehingga tidak terjadi tabrakan. Program ini juga mengatasi masalah *invalid move* yang mungkin terjadi sehingga bot tidak akan berhenti atau *stuck* selama permainan. Hal terakhir yang menjadi prioritas program adalah letak *teleporter*. Keberadaan *teleporter* terkadang dapat merugikan bot, karena bot memilih melewati *teleporter* yang malah menjauhi diamond ataupun base. Oleh karena itu, kami membuat aturan dimana apabila jarak ke diamond lebih dekat daripada jarak ke *teleporter*, maka bot akan memilih rute menuju diamond.

Berikut adalah *sourcecode* strategi *greedy* yang digunakan.

```
### FINAL BOT ###

import random
from typing import Optional, List

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

def calculate_distance(position1, position2):
    return ((position1.x - position2.x) ** 2 + (position1.y - position2.y)
** 2) ** 0.5

@property
def teleport(self) -> List[GameObject]:
    return [d for d in self.game_objects if d.type == "TeleportGameObject"]

class SuperSilverqueen(BaseLogic):
    def __init__(self):
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.goal_position: Optional[Position] = None
        self.current_direction = 0
        self.move_count = 0

    def find_teleport_gameobject(self, board: Board) -> Optional[Position]:
        teleport_objects = [obj for obj in board.game_objects if obj.type ==
"TeleportGameObject"]
        if teleport_objects:
            return teleport_objects[0].position,
```

```

        teleport_objects[1].position
    else:
        return None

    def find_nearest_diamond(self, board_bot: GameObject, board: Board):
        diamonds = board.diamonds
        if diamonds:
            if board_bot.properties.diamonds == 4:
                # Kalau diamonds sudah 4, cari diamond biru
                blue_diamonds = [diamond for diamond in diamonds if
diamond.properties.points != 2]
                if blue_diamonds:
                    nearest_diamond = min(blue_diamonds, key=lambda diamond:
abs(diamond.position.x - board_bot.position.x) + abs(diamond.position.y -
board_bot.position.y))
                else:
                    nearest_diamond = min(diamonds, key=lambda diamond:
abs(diamond.position.x - board_bot.position.x) + abs(diamond.position.y -
board_bot.position.y))
                else:
                    nearest_diamond = min(diamonds, key=lambda diamond:
abs(diamond.position.x - board_bot.position.x) + abs(diamond.position.y -
board_bot.position.y))
            self.goal_position = nearest_diamond.position
        else:
            # No diamonds found, roam around
            self.goal_position = None

    def next_move(self, board_bot: GameObject, board: Board):
        props = board_bot.properties
        bots = board.bots
        # Check other bots position
        for bot in bots:
            if bot != board_bot:
                temp_distance = calculate_distance(bot.position,
board_bot.position)
                if temp_distance <= 1:
                    temp_delta_x = bot.position.x - board_bot.position.x
                    temp_delta_y = bot.position.y - board_bot.position.y
                    # if adjacent, move away
                    if temp_delta_x == 0 and abs(temp_delta_y) == 1:
                        # jika ada bot lawan di sebelah kanan atau kiri,
                        arahkan untuk bergerak ke atas atau bawah
                        if 0 <= board_bot.position.y - temp_delta_y <=
board.width:
                            return 0, -temp_delta_y
                        elif 0 <= board_bot.position.y + temp_delta_y <=
board.width:
                            return 0, temp_delta_y
                    elif temp_delta_y == 0 and abs(temp_delta_x) == 1:
                        # jika ada bot lawan di atas atau bawah, arahkan
                        untuk bergerak ke kanan atau kiri (atur invalid move juga)
                        if 0 <= board_bot.position.x - temp_delta_x <=
board.width:
                            return -temp_delta_x, 0
                        elif 0 <= board_bot.position.x + temp_delta_x <=
board.width:
                            return temp_delta_x, 0
                # lokasi teleport game object

```

```

        teleport_position,    teleport_position1    =
self.find_teleport_gameobject(board)
        if teleport_position:
            print("teleporter location:", teleport_position)
            teleport_x = teleport_position.x
            teleport_y = teleport_position.y
            print("teleporter location - x:", teleport_x, "y:", teleport_y)

        if teleport_position1:
            print("teleporter location:", teleport_position1)
            teleport_x1 = teleport_position1.x
            teleport_y1 = teleport_position1.y
                print("teleporter location - x:", teleport_x1, "y:",
teleport_y1)

# Analyze new state
if props.diamonds == 5:
    # Move to base
    base = board_bot.properties.base
    self.goal_position = base
else:
    # Find the nearest diamond
    self.find_nearest_diamond(board_bot, board)
        if calculate_distance(self.goal_position, board_bot.position) >
calculate_distance(board_bot.properties.base,      board_bot.position) and
props.diamonds >= 3:
            self.goal_position = board_bot.properties.base

current_position = board_bot.position
if self.goal_position:
    # We are aiming for a specific position, calculate delta
    delta_x, delta_y = get_direction(
        current_position.x,
        current_position.y,
        self.goal_position.x,
        self.goal_position.y,
    )
    board_x, board_y = board_bot.position.x, board_bot.position.y
        if (board_x + delta_x == teleport_x and board_y + delta_y ==
teleport_y) or \
            (board_x + delta_x == teleport_x1 and board_y + delta_y ==
teleport_y1):
                distance = calculate_distance(self.goal_position,
board_bot.position)
                    if distance < 8 and (0 <= board_x + delta_x <= 14) and (0 <=
board_y + delta_y <= 14):
:
                return -delta_x, -delta_y
print("current x:", current_position.x)
print("current y:", current_position.y)
print("goal x:", self.goal_position.x)
print("goal y:", self.goal_position.y)
else:
    # Roam around
    delta = self.directions[self.current_direction]
    delta_x = delta[0]
    delta_y = delta[1]
    if random.random() > 0.6:
        self.current_direction = (self.current_direction + 1) % len(
            self.directions

```

```
)  
  
    self.move_count += 1  
    print("Move count:", self.move_count)  
  
    if (0 <= board_bot.position.x + delta_x <= board.width) and (0 <=  
board_bot.position.y + delta_y <= board.width):  
        return delta_x, delta_y  
    else:  
        # adjust movement biar tetap didalam board  
        while (delta_x + board_bot.position.x > board.width) or (delta_x  
+ board_bot.position.x < 0) or (delta_y + board_bot.position.y > board.width) or  
(delta_y + board_bot.position.y < 0):  
            delta_x = random.randint(-1, 1)  
            if delta_x != 0:  
                delta_y = 0  
            else:  
                delta_y = random.choice([-1, 1])  
    return delta_x, delta_y
```

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1.Implementasi dalam Pseudocode

- a. Fungsi `find_teleport_gameobject` :: mencari letak teleporter pada board.

```
Function find_teleport_gameobject(board: Board) -> Position or None:  
    teleport_objects <- []  
    for each obj in board.game_objects:  
        if obj.type = "TeleportGameObject" then  
            add obj to teleport_objects  
        endif  
    endfor  
  
    if teleport_objects ≠ empty then  
        → teleport_objects[0].position, teleport_objects[1].position  
    else  
        → None  
    endif  
endfunction
```

- b. Algoritma `find_nearest_diamond` :: mencari diamond terdekat dari posisi bot pemain.

```
Function find_nearest_diamond(board_bot: GameObject, board: Board):  
    diamonds <- board.diamonds  
    if diamonds ≠ empty then  
        if board_bot.properties.diamonds = 4 then  
            blue_diamonds = []  
            for each diamond in diamonds:  
                if diamond.properties.points ≠ 2 then  
                    blue_diamonds <- blue_diamonds + [diamond]  
                endif  
            endfor  
  
            if blue_diamonds ≠ empty then  
                nearest_diamond <- min(blue_diamonds, key=lambda  
diamond: abs(diamond.position.x - board_bot.position.x) +  
abs(diamond.position.y - board_bot.position.y))  
            else  
                nearest_diamond <- min(diamonds, key=lambda diamond:  
abs(diamond.position.x - board_bot.position.x) + abs(diamond.position.y -  
board_bot.position.y))  
            endif  
        else  
            nearest_diamond <- min(diamonds, key=lambda diamond:  
abs(diamond.position.x - board_bot.position.x) + abs(diamond.position.y -  
board_bot.position.y))  
        endif  
    endif
```

```

        endif
        board_bot.goal_position ← nearest_diamond.position
    else
        { No diamonds found, roam around }
        board_bot.goal_position ← None
    endif
endfunction

```

- c. Algoritma next_move :: mencari langkah selanjutnya dari bot pemain dengan mempertimbangkan bot lawan serta invalid move.

```

Function next_move(self, board_bot: GameObject, board: Board):
    props ← board_bot.properties
    bots ← board.bots

    { Check for nearby bots and move away if needed }
    for each bot in bots:
        if bot ≠ board_bot then
            temp_distance ← calculate_distance(bot.position,
board_bot.position)
            if temp_distance ≤ 1 then
                temp_delta_x ← bot.position.x - board_bot.position.x
                temp_delta_y ← bot.position.y - board_bot.position.y

                { Move away from adjacent bot }
                if temp_delta_x = 0 and abs(temp_delta_y) = 1 then
                    if 0 ≤ board_bot.position.y - temp_delta_y ≤
board.width then
                        Return 0, -temp_delta_y
                    elseif 0 ≤ board_bot.position.y + temp_delta_y ≤
board.width then
                        Return 0, temp_delta_y
                    endif
                elseif temp_delta_y = 0 and abs(temp_delta_x) = 1 then
                    if 0 ≤ board_bot.position.x - temp_delta_x ≤
board.width then
                        Return -temp_delta_x, 0
                    elseif 0 ≤ board_bot.position.x + temp_delta_x ≤
board.width then
                        Return temp_delta_x, 0
                    endif
                endif
            endif
        endif
    endfor

    { Cari lokasi teleporter }
    teleport_position,      teleport_position1 ←
self.find_teleport_gameobject(board)
    if teleport_position then

```

```

        teleport_x <- teleport_position.x
        teleport_y <- teleport_position.y
    endif
    if teleport_position1 then
        teleport_x1 <- teleport_position1.x
        teleport_y1 <- teleport_position1.y
    endif

    { Analyze new state and determine movement }
    if props.diamonds = 5 then
        { Move to base }
        base <- board_bot.properties.base
        self.goal_position <- base
    else
        { Find nearest diamond }
        self.find_nearest_diamond(board_bot, board)
        if calculate_distance(self.goal_position, board_bot.position)
> calculate_distance(board_bot.properties.base, board_bot.position) and
props.diamonds ≥ 3 then
            self.goal_position <- board_bot.properties.base
        endif
    endif

    current_position <- board_bot.position
    if self.goal_position then
        { Calculate delta for specific position }
        delta_x, delta_y <- get_direction(current_position.x,
current_position.y, self.goal_position.x, self.goal_position.y)
        board_x, board_y <- board_bot.position.x, board_bot.position.y

        { Check if next move is a teleport position }
        if (board_x + delta_x = teleport_x and board_y + delta_y =
teleport_y) or (board_x + delta_x = teleport_x1 and board_y + delta_y =
teleport_y1) then
            distance <- calculate_distance(self.goal_position,
board_bot.position)
            if distance < 8 and (0 ≤ board_x + delta_x ≤ 14) and (0
≤ board_y + delta_y ≤ 14) then
                → -delta_x, -delta_y
            endif
        endif
    else
        { Roam around }
        delta <- self.directions[self.current_direction]
        delta_x <- delta[0]
        delta_y <- delta[1]
        if random.random() > 0.6 then
            self.current_direction <- (self.current_direction + 1) %
len(self.directions)
        endif
    endif

```

```

        self.move_count += 1

        { Adjust movement to stay within board boundaries }
        if (0 ≤ board_bot.position.x + delta_x ≤ board.width) and (0 ≤
            board_bot.position.y + delta_y ≤ board.width) then
            → delta_x, delta_y
        else
            while (delta_x + board_bot.position.x > board.width) or
                (delta_x + board_bot.position.x < 0) or (delta_y + board_bot.position.y >
                    board.width) or (delta_y + board_bot.position.y < 0) Do
                delta_x ← random.randint(-1, 1)
                if delta_x ≠ 0 then
                    delta_y ← 0
                else
                    delta_y ← random.choice([-1, 1])
                endif
            endwhile
            → delta_x, delta_y
        endif
    endfunction

```

4.2. Struktur Data Program

Struktur program dari permainan Diamonds sudah tersedia di dalam starter-bots yang diberikan bersamaan spesifikasi tugas ini, yang diantaranya terdiri dari beberapa bagian utama, yaitu models, util, base, dan main.

a. Models

Pada file models.py ini, didefinisikan kelas-kelas model permainan, diantaranya:

1. Bot: Merepresentasikan data dari seorang bot, dengan atribut name, email, dan id dari bot.
2. Position: Mendefinisikan suatu posisi dengan koordinat x dan y.
3. Base: Subkelas dari Position, merepresentasikan posisi basis (*base*).
4. Properties: Menyimpan atribut-atribut properti seperti points, pair_id, diamonds, score, name, inventory_size, can_tackle, milliseconds_left, time_joined, dan base.
5. GameObject: Objek dalam permainan dengan atribut id, position, type, dan properties.

6. Config: Konfigurasi permainan dengan atribut seperti generation_ratio, min_ratio_for_generation, red_ratio, seconds, pairs, inventory_size, dan can_tackle.
7. Feature: Fitur dalam permainan dengan atribut name dan config.
8. Board: Papan permainan dengan atribut id, width, height, features, minimum_delay_between_moves, dan game_objects. Memiliki metode bots untuk mendapatkan semua objek bot dalam papan, diamonds untuk mendapatkan semua objek *diamond*, dan get_bot untuk mendapatkan objek bot berdasarkan nama bot. Metode is_valid_move digunakan untuk memeriksa apakah gerakan yang diminta oleh bot adalah valid.

b. Util

Kode yang terdapat dalam file util.py ini terdiri dari fungsi-fungsi utilitas yang digunakan untuk mengolah posisi dan arah dalam permainan, seperti pada berikut.

1. clamp(n, smallest, largest): Fungsi ini mengembalikan nilai n yang dibatasi oleh smallest dan largest.
2. get_direction(current_x, current_y, dest_x, dest_y): Fungsi ini mengembalikan arah delta (delta_x, delta_y) dari posisi saat ini (current_x, current_y) ke posisi tujuan (dest_x, dest_y).
3. position_equals(a: Position, b: Position): Fungsi ini memeriksa apakah dua objek Position sama, yaitu jika koordinat x dan y-nya sama.

c. Base

File base.py mendefinisikan kelas BaseLogic yang merupakan kelas abstrak dan memiliki satu metode abstrak next_move. Di sini, kelas merupakan bagian dari *logic* yang bertindak sebagai dasar untuk logika permainan yang akan diimplementasikan oleh kelas turunannya. Kelas BaseLogic memiliki metode sebagai berikut.

1. next_move(board_bot: GameObject, board: Board) -> Tuple[int, int]: Metode abstrak yang harus diimplementasikan oleh kelas turunan. Metode ini menerima objek board_bot yang merupakan objek dalam permainan dan board yang merupakan papan permainan, dan mengembalikan

tuple berisi dua nilai integer yang merepresentasikan langkah berikutnya yang akan diambil oleh objek `board_bot`.

d. Main

File `main.py` ini berisi inisialisasi bot saat dipanggil dari file executable `.bat` atau `.sh`. Kode-kode yang terdapat dalam file ini mengatur jalannya permainan serta memanipulasi objek-objek dalam permainan sesuai dengan aturan dan logika yang ditentukan. Program akan melakukan loop selama permainan berlangsung serta mendapatkan informasi bot dari papan permainan. Selanjutnya, ia akan menghitung langkah berikutnya menggunakan logika yang telah ditentukan. Program juga akan melakukan validasi gerakan yang diminta dan melakukan gerakan serta mendapatkan pembaruan status papan permainan. Jika permainan berakhir, program akan keluar dari loop dan mencetak pesan "Game over!" saat permainan berakhir.

Dalam proses pembuatan bot, dibuat juga sebuah kelas baru bernama `SuperSilverqueen` yang merupakan turunan dari kelas `BaseLogic` pada `base.py` dengan:

a. Atribut

1. `directions`: Sebuah list yang berisi tuple-tuple yang merepresentasikan arah gerakan (kanan, atas, kiri, bawah).
2. `goal_position`: Posisi yang menjadi tujuan robot.
3. `current_direction`: Indeks yang menunjukkan arah saat ini dari list `directions`.
4. `move_count`: Jumlah langkah yang telah diambil oleh robot.

b. Metode

1. `find_teleport_gameobject(board)`: Metode ini mencari objek `teleporter` di papan permainan (`board`) dan mengembalikan posisi teleportasi tersebut jika ada, atau `None` jika tidak ada.
2. `find_nearest_diamond(board_bot, board)`: Metode ini mencari diamond terdekat dari posisi robot di papan permainan (`board`) dan menetapkan posisi tersebut sebagai tujuan (`goal_position`) robot.
3. `next_move(board_bot, board)`: Metode utama yang menentukan langkah selanjutnya yang akan diambil oleh robot berdasarkan kondisi saat ini. Metode

ini memperhitungkan posisi bot lain, posisi teleportasi, dan tujuan robot (baik diamond terdekat atau base).

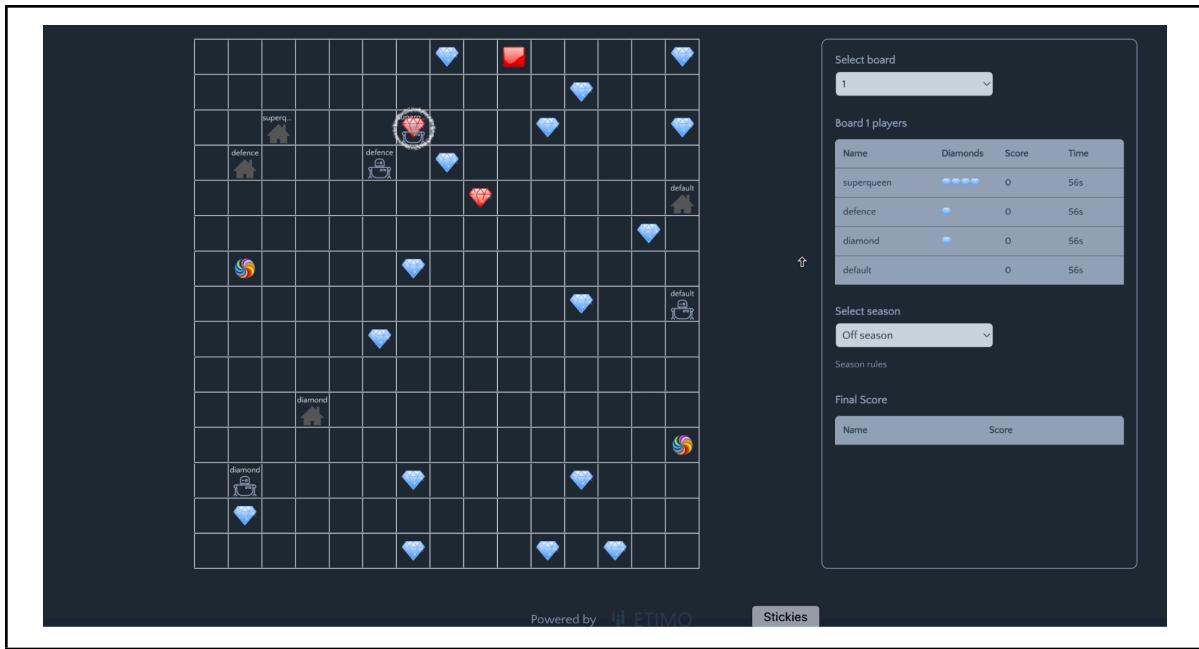
Selain itu, dibuat pula fungsi baru bernama `calculate_distance(position1, position2)` yang dimanfaatkan oleh kelas SuperSilverqueen dalam menghitung jarak dari bot pemain ke diamond, base, *teleporter*, ataupun bot lawan.

4.3. Analisis dan Pengujian

Analisis dan pengujian dilakukan dengan cara menjalankan permainan dengan memasukkan bot kami (SuperSilverqueen) dengan beberapa *reference bot* yang telah sedikit kami kembangkan. Bot ‘default’ bergerak dengan logika *reference bot* yang sama sekali tidak kami ubah, bot ‘diamond’ bergerak dengan logika *greedy by diamond* saja, bot ‘defence’ bergerak dengan logika *greedy by defence* saja, dan bot ‘superqueen’ bergerak dengan gabungan logika *greedy by distance, diamond, dan defence*.

Berikut adalah beberapa analisis bot yang dilakukan dalam beberapa pertandingan.

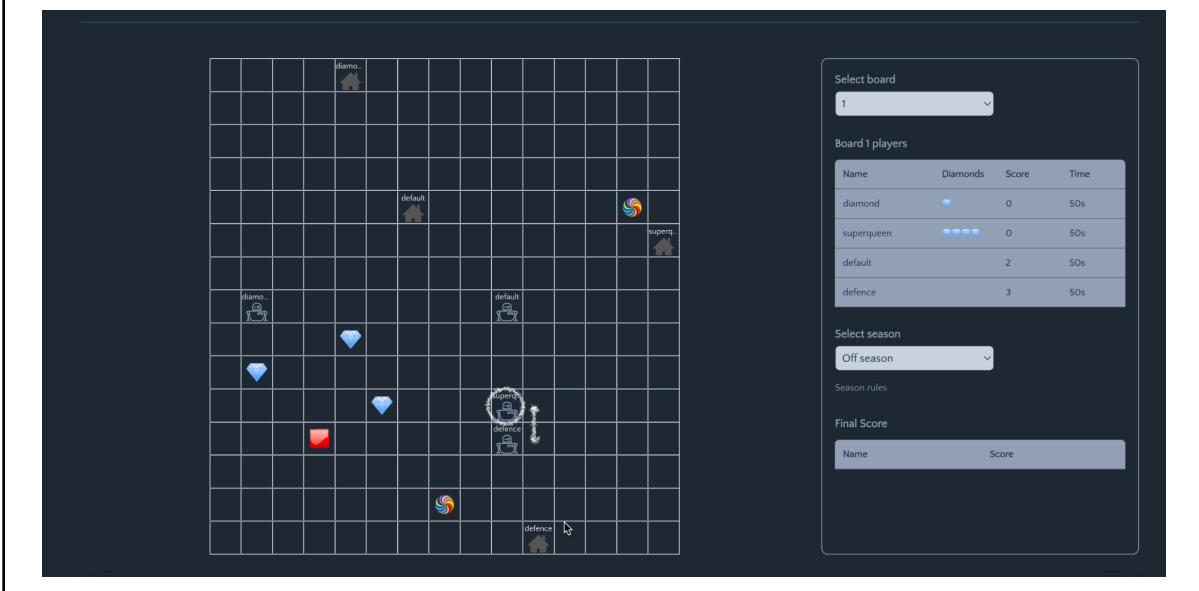
Pengambilan Diamond
Implementasi Strategi: Strategi diimplementasikan dengan mengutamakan diamond yang jaraknya paling dekat, namun apabila diamond merah memiliki jarak yang sama dengan diamond biru, maka pengambilan diamond merah diutamakan. Strategi juga membuat bot tidak lagi mengambil diamond merah yang memiliki point 2 apabila count diamond saat itu sudah mencapai 4.
Hasil: Bot ‘superqueen’ sudah efektif dalam mengambil diamond dan tidak lagi mengambil diamond merah saat count diamond sudah 4, melainkan ia akan mengambil diamond biru (<i>lihat gambar</i>).

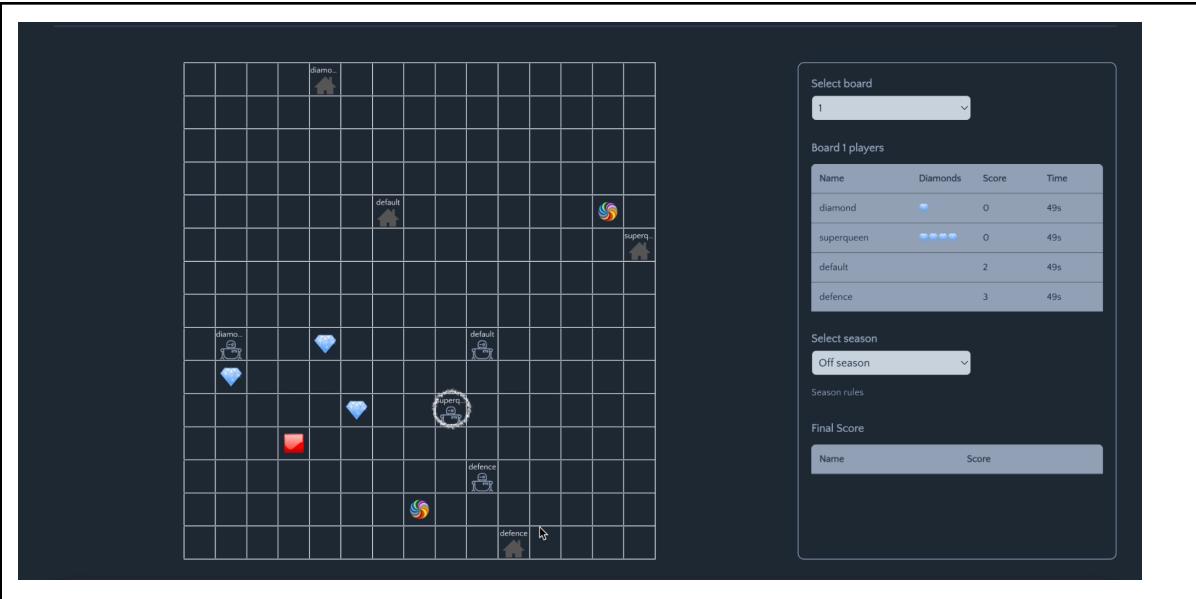


Menjauhi Bot Lawan

Implementasi Strategi: Strategi diimplementasikan dengan membuat bot bergerak kearah yang menjauhi lawan apabila lawan terletak sejajar dengan bot pemain agar tidak terjadi tabrakan yang membuat kehilangan diamond.

Hasil: Bot ‘superqueen’ sudah efektif dalam menjauhi bot lawan, yang pada kasus ini adalah bot ‘defence’ (*lihat gambar*).

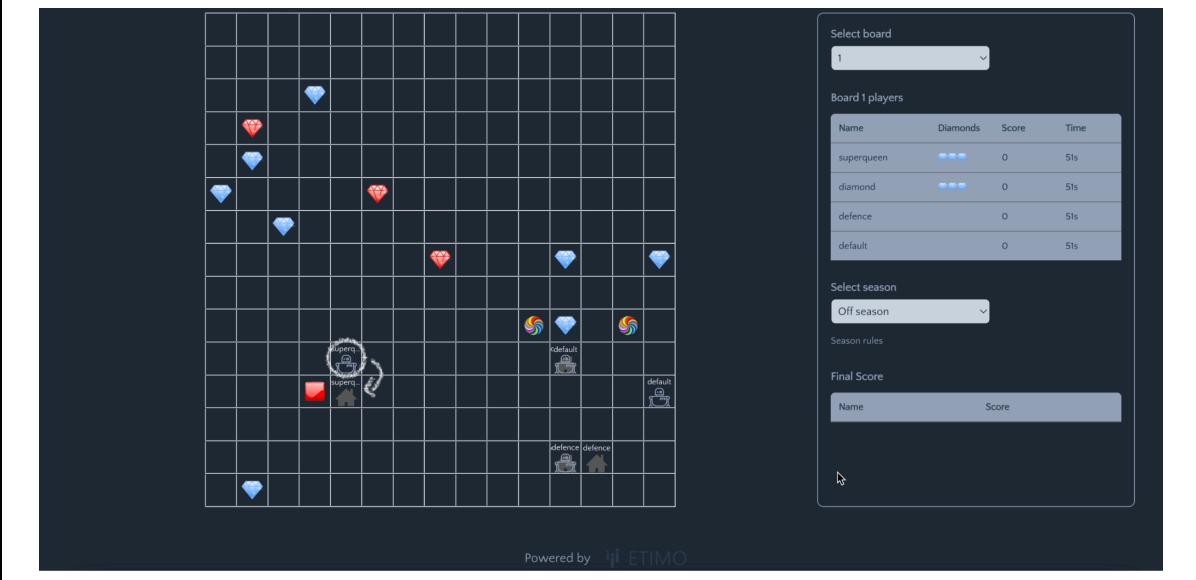




Return to Base

Implementasi Strategi: Strategi diimplementasikan dengan membuat bot bergerak menuju base apabila diamond count sudah lebih atau sama dengan 3 dan jarak base lebih dekat dibanding diamond selanjutnya.

Hasil: Bot ‘superqueen’ sudah efektif dalam memilih untuk menyimpan diamond (pulang ke base) daripada mengambil diamond yang jauh.



BAB 5

PENUTUP

5.1.Kesimpulan

Setelah melakukan berbagai eksplorasi terkait algoritma *greedy* dan implementasinya pada permainan Diamonds, kami menyadari bahwa algoritma *greedy* merupakan algoritma yang cukup baik dan optimal dalam meningkatkan efisiensi dan optimasi permainan Diamonds. Contohnya adalah pada pengambilan diamond, penentuan jarak terdekat, dan penghindaran dari lawan. Program akhir yang kami buat merupakan hasil dari penggabungan beberapa solusi *greedy* yang telah disebutkan sebelumnya dan telah diuji untuk menghasilkan strategi terbaik menurut penulis.

5.2.Saran

Kami memiliki beberapa saran dalam penggerjaan proyek yang sejenis dengan tugas besar ini:

1. Memahami dengan baik struktur data permainan, seperti game-engine, karena terdapat beberapa isu yang tidak bisa diselesaikan hanya dengan memodifikasi bot.
2. Lokasi bot pada *board* pada akhirnya sebenarnya sangat bergantung pada tingkat keberuntungan, jadi tidak usah terlalu stres dalam membangun algoritma.

5.3.Refleksi

Tugas besar ini telah memberikan pengalaman yang berharga bagi kami atas situasi dan tantangan yang muncul. Salah satu kendala yang kami hadapi adalah pembagian waktu dan porsi penggerjaan yang kurang efektif dan efisien. Namun, pada akhirnya, tugas besar ini membantu meningkatkan kemampuan problem solving, computational thinking, dekomposisi masalah, komunikasi, serta kerjasama kami. Harapannya, semoga strategi algoritma yang telah dibuat dapat dikembangkan kembali sehingga efektivitasnya dapat meningkat.

DAFTAR PUSTAKA

- Munir, R. “Algoritma Greedy (Bag. 1)”, [Online]. Available: [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). [Diakses 4 Maret 2024].
- Munir, R. “Algoritma Greedy (Bag. 2)”, [Online]. Available: [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). [Diakses 4 Maret 2024].
- Munir, R. “Algoritma Greedy (Bag. 3)”, [Online]. Available: [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). [Diakses 4 Maret 2024].
- Shiba, N. “Pengertian Game Engine, Jenis dan Fungsinya”, [Online]. Available: <https://ids.ac.id/pengertian-game-engine-jenis-dan-fungsinya/>. [Diakses 7 Maret 2024].

LAMPIRAN

LINK REPOSITORY

Link Github *Repository* Tugas Besar ini dapat diakses pada tautan:
https://github.com/denoseu/Tubes1_sem-4-dapat-silverqueen.semoga

LINK VIDEO YOUTUBE

Link video Youtube Tugas Besar ini dapat diakses pada tautan: <https://youtu.be/HhVUKvQr8nY>.

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522013	Denise Felicia Tiowanni	Pengembangan algoritma <i>greedy by diamond</i> , penggabungan algoritma secara keseluruhan, pembuatan video, penulisan laporan.
13522063	Shazya Audrea Taufik	Pengembangan algoritma <i>greedy by defence</i> , penggabungan algoritma secara keseluruhan, pembuatan video, penulisan laporan.
13522087	Shulha	Pengembangan algoritma <i>greedy by distance</i> , penggabungan algoritma secara keseluruhan, pembuatan video, penulisan laporan.