IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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

Oleh: Kelompok 5 (2econd)

Ribka Hana Josephine Situmorang	123140103
Sahiva Syamdo Vinoza	123140194

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

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

Spesifikasi Tubes

- Buatlah program sederhana dalam bahasa Python yang mengimplementasikan algoritma Greedy pada bot permainan Diamonds dengan tujuan memenangkan permainan.
- Tugas dikerjakan berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang, boleh lintas kelas dan lintas kampus.
- Buatlah strategi greedy terbaik, karena setiap bot dari masing-masing kelompok akan diadu dalam kompetisi Tubes 1. Tiap kelompok dapat menggunakan kreativitas yang bermacam-macam dalam menyusun strategi greedy untuk menang.
- Jika terdapat kesulitan selama mengerjakan tugas besar sehingga memerlukan bimbingan, maka dapat melakukan asistensi tugas besar kepada asisten (opsional).
- Bot yang telah dibuat akan dikompetisikan dengan kelompok lain dan disaksikan oleh seluruh peserta kuliah. Terdapat hadiah menarik bagi kelompok yang memenangkan kompetisi.
- Kelompok yang terindikasi melakukan kecurangan akan diberikan nilai 0 pada tugas besar.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma greedy merupakan metode yang paling populer dan sedehana untuk memecahkan persoalan optimasi. Algoritma greedy memecahkan persoalan secara langkah per langkah sedemikian sehingga, pada setiap langkah mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "take what you can get now!") dan "berharap" bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Kegunaan utama dari algoritma greedy adalah untuk menemukan solusi optimal dalam persoalan optimasi dengan cepat.

Berdasarkan buku Pengantar Strategi Algoritma, Gia S. Wulandari dan Siti Saadah, 2021:56), algoritma greedy biasanya digunakan untuk menyelesaikan permasalahan optimasi. Permasalahan optimasi sendiri merupakan permasalahan yang menuntut pencarian solusi optimum dari sekumpulan alternatif solusi yang mungkin. Dengan selalu memilih optimum lokal, maka diharapkan untuk dapat mencapai optimum global. Secara umum, terdapat enam komponen yang ada pada algoritma greedy, yaitu:

1. Himpunan kandidat, merupakan himpunan yang berisi objek-objek yang dapat dipilih pada langkah-langkah algoritma greedy.
2. Himpunan solusi, merupakan himpunan yang berisi, atau merepresentasikan, objek-objek yang telah terpilih pada langkah-langkah algoritma greedy.
3. Fungsi solusi, merupakan fungsi yang mengindikasikan bahwa kita telah menemukan solusi yang lengkap. Fungsi inilah yang menjadi penentu akhir dari suatu iterasi.
4. Fungsi seleksi, menyatakan aturan dasar pemilihan suatu kandidat untuk dijadikan sebagai bagian dari solusi.
5. Fungsi kelayakan, fungsi yang menetapkan apakah suatu kandidat dapat masuk ke dalam solusi berdasarkan kendala yang ada.
6. Fungsi objektif, fungsi yang meminimalkan atau memaksimalkan suatu nilai, yang menjadi tujuan utama dari masalah terkait.

2.2 Cara Kerja Program

Program bot ini beroperasi sebagai sebagai agen yang menerima data kondisi terkini dari game engine melalui API secara terus-menerus dan menganalisis informasi menggunakan logika internal yang sudah terprogram untuk mengambil keputusan. Setiap keputusan yang akan menghasilkan opsi yang paling menguntungkan secara local berdasarkan kriteria.

1. **Cara Implementasi Program**

   Implementasi program ini dari algoritma greedy terhadap program bot, meliputi beberapa hal sebagai berikut:

   Langkah pertama yang dilakukan adalah Inisialisasi ketika bot pertama kali dibuat.

```python
class BotGakLogis(BaseLogic):
    def __init__(self):
        super().__init__()
        self.goal_position = None
        self.previous_position = (None, None)
        self.turn_direction = 1
        self.stuck_counter = 0
        self.max_stuck_attempts = 3
        self.tackle_threshold_diamonds = 3
        self.targeted_enemy_id = None
        self.scan_radius = 5
        self.exploration_direction = random.choice([(0, 1), (0, -1), (1, 0), (-1, 0)])
        self.red_diamond_score_multiplier = 2.0
```

   Langkah Kedua, mengambil keputusan pada setiap giliran dan bot akan menjalankan fungsi untuk menentukan langkah yang diambil berdasarkan kondisi.

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

   Jika inventory penuh, bot akan kembali ke base

```python
if props.diamonds >= props.inventory_size:
    self.goal_position = props.base
    self.targeted_enemy_id = None # Reset target
    if position_equals(current_position, self.goal_position):
        self.goal_position = None
```

Jika waktu sudah mau habis, bot akan kembali ke base

```
# Jika waktu sudah mau habis, bot akan kembali ke base
elif hasattr(board, 'time_left') and board.time_left < 10000 and props.diamonds > 0:
    self.goal_position = props.base
    self.targeted_enemy_id = None
    if position_equals(current_position, self.goal_position):
        self.goal_position = None
```

Mencari diamond, yang diprioritaskan adalah redDiamond yang bernilai 2

```
else:
    best_diamond = self._find_best_diamond_in_radius(current_position, props, board)
    if best_diamond:
        self.goal_position = best_diamond.position
        self.targeted_enemy_id = None
        self.exploration_direction = random.choice([(0, 1), (0, -1), (1, 0), (-1, 0)])
    else:
```

Jika tidak ada diamond dalam radius scan, baru cari bot musuh

```
tackle_target = self._find_tackle_target_in_radius(board_bot, board)
if tackle_target:
    self.goal_position = tackle_target.position
    self.targeted_enemy_id = tackle_target.id
    self.exploration_direction = random.choice([(0, 1), (0, -1), (1, 0), (-1, 0)])
else:
```

Jika tidak ada diamond dan tidak ada bot musuh dalam radius, bot akan kembali ke base

```
if props.diamonds > 0:
    self.goal_position = props.base
    self.targeted_enemy_id = None
    self.exploration_direction = random.choice([(0, 1), (0, -1), (1, 0), (-1, 0)])
else:
    self.goal_position = None
    self.targeted_enemy_id = None
```

bergerak ke arah tujuan.

```
if self.goal_position:
    return self._move_towards_goal(current_position, board)
else:
    return self._semi_random_explore(current_position, board)
```

Bot akan mencari berlian terbaik, dalam radius tertentu.

```
def _find_best_diamond_in_radius(self, current_position, props, board):  1 usage
    best_score = -1.0
    best_diamond = None
    space_left = props.inventory_size - props.diamonds
```

Pastikan diamond masih muat dan dalam radius penglihatan.

```
for diamond in board.diamonds:
    if (diamond.properties.points <= space_left and
        not position_equals(current_position, diamond.position) and
        self._is_in_scan_radius(current_position, diamond.position)):

        distance = self._distance(current_position, diamond.position)

        diamond_value = diamond.properties.points
```

Jika red diamond, nilai dikali multiplier. Skor = nilai / (jarak + 1)

```
        diamond_value = diamond.properties.points
        if hasattr(diamond.properties, 'type') and diamond.properties.type == 'redDiamond':
            diamond_value *= self.red_diamond_score_multiplier

        current_score = diamond_value / (distance + 1)

        if current_score > best_score:
            best_score = current_score
            best_diamond = diamond

return best_diamond
```

Hitung jarak

```
def _distance(self, pos1, pos2):  2 usages
    return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)
```

Cek apakah posisi tidak berubah (stuck)

```
def _move_towards_goal(self, current_position, board):  1 usage
    cur_x, cur_y = current_position.x, current_position.y

    if (cur_x, cur_y) == self.previous_position:
        self.stuck_counter += 1
    else:
        self.stuck_counter = 0
```

Jika stuck terlalu lama, reset tujuan dan bergerak acak

```
if self.stuck_counter >= self.max_stuck_attempts:
    self.stuck_counter = 0
    self.goal_position = None
    return self._random_move()
```

Hitung arah menuju goal

```
delta_x, delta_y = get_direction(
    cur_x, cur_y,
    self.goal_position.x, self.goal_position.y
)
```

Jika arah tidak berubah tapi tetap stuck, coba belok

```
if (cur_x, cur_y) == self.previous_position and (delta_x != 0 or delta_y != 0):
    if delta_x != 0:
        delta_y = delta_x * self.turn_direction
        delta_x = 0
    elif delta_y != 0:
        delta_x = delta_y * self.turn_direction
        delta_y = 0

    self.turn_direction *= -1

if not board.is_valid_move(current_position, delta_x, delta_y):
    return self._random_move()
```

Jika arah tidak valid (terhalang), bergerak acak

```
if not board.is_valid_move(current_position, delta_x, delta_y):
    return self._random_move()

self.previous_position = (cur_x, cur_y)
return delta_x, delta_y
```

Bergerak ke arah acak (up/down/left/right)

```
def _random_move(self):  2 usages
    moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]
    return random.choice(moves)
```

Coba bergerak ke arah eksplorasi saat ini

```
def _semi_random_explore(self, current_position, board):  1 usage
    delta_x, delta_y = self.exploration_direction

    if board.is_valid_move(current_position, delta_x, delta_y):
        self.previous_position = (current_position.x, current_position.y)
        return delta_x, delta_y
    else:
```

Jika arah eksplorasi terhalang, cari arah lain secara acak

```
valid_moves = []
for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
    if board.is_valid_move(current_position, dx, dy):
        valid_moves.append((dx, dy))

if valid_moves:
    new_direction = random.choice(valid_moves)
    self.exploration_direction = new_direction
    self.previous_position = (current_position.x, current_position.y)
    return new_direction
else:
    return 0, 0
```

Cek apakah target sebelumnya masih valid

```
def _find_tackle_target_in_radius(self, board_bot: GameObject, board: Board):  1 usage
    if self.targeted_enemy_id:
        for bot in board.game_objects:
            if bot.type == "BotGameObject" and bot.id == self.targeted_enemy_id:
                # Jika target masih valid dan punya cukup berlian dan dalam radius, target akan dikejar
                if (bot.properties.diamonds >= self.tackle_threshold_diamonds and
                    self._is_in_scan_radius(board_bot.position, bot.position)):
                    return bot
                else:
                    self.targeted_enemy_id = None
                    break

    best_target = None
    max_diamonds = self.tackle_threshold_diamonds - 1
```

Cari musuh dengan diamond terbanyak di radius

```
for bot in board.game_objects:
    if bot.type == "BotGameObject" and bot.id != board_bot.id:
        if (bot.properties.diamonds > max_diamonds and
            self._is_in_scan_radius(board_bot.position, bot.position)):
            max_diamonds = bot.properties.diamonds
            best_target = bot

if best_target and best_target.properties.diamonds >= self.tackle_threshold_diamonds:
    return best_target

return None
```

Cek apakah posisi berada dalam radius penglihatan

```
def _is_in_scan_radius(self, pos1, pos2):  3 usages
    return self._distance(pos1, pos2) <= self.scan_radius
```

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Pendekatan Greedy dalam persoalan Diamonds harus dilakukan pemetaan elemen elemen masalah yaitu :

3.1.1 Himpunan Kandidat

Himpunan kandidat merupakan semua tujuan yang dapat dipilih oleh program dalam setiap langkahnya, dalam aplikasi ini himpunan kandidatnya berupa :

1. Diamond
2. Bot Musuh
3. Teleporter
4. Base

3.1.2 Himpunan Solusi

Himpunan solusi merupakan langkah yang diambil bot untuk mencapai tujuan, dalam game ini tujuannya adalah mengumpulkan diamond sebanyak-banyaknya. Dalam aplikasi ini, himpunan solusinya berupa :

1. Posisi diamond terdekat apabila slot inventory masih tersedia.
2. Posisi bot musuh apabila ada musuh.
3. Posisi teleporter apakah mempercepat langkah kembali ke base atau langkah ke diamond terdekat.
4. Posisi base apabila inventory sudah penuh atau waktu sudah mau habis.

3.1.3 Fungsi Solusi

Fungsi solusi merupakan fungsi yang menentukan kapan suatu solusi dianggap tuntas. Dalam aplikasi ini fungsi solusinya adalah :

1. Inventory sudah penuh dan bot kembali ke base.
2. Waktu permainan habis.

3.1.4 Fungsi Seleksi

Fungsi seleksi merupakan fungsi yang menjelaskan mengapa langkah yang diambil bot adalah langkah yang paling efektif, dalam aplikasi ini fungsi seleksinya adalah :

1. Bot mencari diamond terdekat berdasarkan rumus *distance*.
2. Bot menghitung jarak ke bot musuh dan mengejar bot musuh apabila jarak dekat.
3. Bot menuju teleporter terdekat apabila teleporter mempercepat langkah kembali ke base atau mempercepat langkah ke diamond terdekat.
4. Bot kembali ke base, dapat melalui teleporter atau tidak, tergantung dari jarak base.

3.1.5 Fungsi Kelayakan

Fungsi kelayakan menentukan apakah suatu kandidat yang dipilih dapat ditambahkan ke himpunan solusi parsial yang sedang dibangun. Fungsi ini memeriksa apakah sebuah langkah atau pilihan memenuhi syarat-syarat tertentu pada kondisi saat itu. Dalam aplikasi ini fungsi kelayakannya adalah :

1. Teleporter mempersingkat langkah menuju tujuan (base maupun diamond)
2. Posisi bot musuh terlalu dekat.
3. Posisi tujuan belum diambil bot lain.

3.1.6 Fungsi Objektif

Fungsi objektif merupakan tujuan akhir dari algoritma yang ingin dicapai melalui serangkaian langkah-langkah yang diambil. Dalam aplikasi ini, fungsi objektifnya adalah :

1. Menggunakan langkah tercepat untuk menuju diamond, base dan teleporter.
2. Kembali ke base setiap inventory penuh agar bot dapat mengumpulkan diamond kembali.
3. Kembali ke base menggunakan teleporter apabila mempersingkat langkah.
4. Kembali ke base dengan biasa apabila teleporter tidak mempersingkat langkah.

3.2 Eksplorasi Alternatif Solusi Greedy

Ada beberapa pendekatan yang dapat digunakan untuk membuat bot yang mengaplikasikan algoritma greedy, seperti :

1. Greedy by Distance

Bot akan selalu memilih diamond dengan jarak terdekat tanpa memperhatikan nilai diamond maupun bot lain.

2. Greedy by Points

   Bot akan selalu memilih diamond dengan poin tertinggi, dalam kasus ini diamond bernilai 2 poin tanpa memperhatikan jarak dan bot musuh.

3. Greedy by Value

   Bot akan menghitung value diamond dengan membagi jumlah poin dengan jarak tempuh. Bot akan memprioritaskan diamond dengan value terbesar.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setiap solusi greedy memiliki keuntungan nya masing-masing dalam pengaplikasiannya, yaitu :

1. Greedy by Distance

   Efisiensi :

   Sangat efisien, bot hanya menghitung jarak dari setiap diamond dalam radius dan memilih jarak yang terkecil.

   Efektivitas :

   Efektivitas sedang, bot akan selalu bergerak untuk mengumpulkan diamond dan tidak akan mengejar diamond dengan poin 2 namun jaraknya jauh. Namun karena algoritma ini mengabaikan nilai, maka poin tidak akan maksimal.

2. Greedy by Points

   Efisiensi :

   Efisien, bot akan mengumpulkan diamond dengan nilai tertinggi dulu, dalam kasus ini 2 poin dan mengabaikan yang diamond berpoin 1.

   Efektivitas :

   Efektivitas sedang, bot akan selalu memprioritaskan diamond yang bernilai tinggi yang bagus untuk kumulatif poin, namun buruknya bot tidak memperhitungkan jarak sehingga bot memprioritaskan diamond berpoin 2 yang berjarak jauh dibanding diamond dengan poin 1 yang lebih dekat.

3. Greedy by Value

   Efisiensi :

   Sangat efisien, bot ini merupakan gabungan dari 2 pendekatan greedy diatas, dimana diamond yang diprioritaskan adalah diamond yang memiliki value terbesar, yang berarti diamond yang memiliki poin tertinggi dan jarak yang dekat.

   Efektivitas :

   Lebih efektif dibanding 2 pendekatan greedy diatas karna bot mempertimbangkan dua aspek penting dalam game ini yaitu poin dan waktu, tidak hanya fokus ke salah satu saja.

3.4 Strategi Greedy yang Dipilih

Strategi yang kami gunakan dalam bot kami adalah pengembangan dari pendekatan algoritma Greedy by Value. Dimana bot akan memilih diamond dengan jarak terdekat dan poin terbesar untuk dikumpulkan duluan.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
Unset
START

  INIT
    goal_position = None
    previous_position = (None, None)
    turn_direction = 1
    stuck_counter = 0
    max_stuck_attempts = 3
    tackle_threshold_diamonds = 3
    targeted_enemy_id = None
    SCAN_RADIUS = 5
    exploration_direction = random direction (up, down, left, right)

  FUNCTION next_move(current_position, properties, board)

    IF inventory penuh THEN
      goal_position = base
      targeted_enemy_id = None
      IF current_position == goal_position THEN
        goal_position = None

    ELSE IF waktu hampir habis AND punya berlian THEN
      goal_position = base
      targeted_enemy_id = None
      IF current_position == goal_position THEN
        goal_position = None

    ELSE
      best_diamond = cari diamond terbaik dalam radius SCAN_RADIUS
      IF best_diamond ada THEN
        goal_position = posisi best_diamond
        targeted_enemy_id = None
```

```
      reset arah eksplorasi random

    ELSE
      target_musuh = cari bot musuh yang bisa diserang dalam radius
      IF target_musuh ada THEN
        goal_position = posisi musuh
        targeted_enemy_id = id musuh
        reset arah eksplorasi random

      ELSE
        IF punya berlian THEN
          goal_position = base
          targeted_enemy_id = None
          reset arah eksplorasi random
        ELSE
          goal_position = None
          targeted_enemy_id = None
        ENDIF
      ENDIF
    ENDIF
  ENDIF

  IF goal_position ada THEN
    return gerak_towards_goal(current_position, board)
  ELSE
    return gerak_semi_random(current_position, board)
  ENDIF

END FUNCTION


FUNCTION gerak_towards_goal(current_position, board)
  IF posisi sama dengan previous_position THEN
    stuck_counter += 1
  ELSE
    stuck_counter = 0

  IF stuck_counter >= max_stuck_attempts THEN
    stuck_counter = 0
    goal_position = None
```

```
    return gerak_random()

  arah = hitung arah menuju goal_position

  IF posisi sama dengan previous_position THEN
    coba gerak alternatif (perpendicular)
    ubah arah giliran

  IF gerak tidak valid di board THEN
    return gerak_random()

  previous_position = current_position
  return arah

END FUNCTION


FUNCTION gerak_random()
  pilih gerak acak dari [atas, bawah, kanan, kiri]
  return gerak

END FUNCTION


FUNCTION gerak_semi_random(current_position, board)
  coba gerak sesuai exploration_direction
  IF gerak valid THEN
    previous_position = current_position
    return gerak
  ELSE
    cari gerak valid lain secara acak
    IF ada gerak valid lain THEN
      update exploration_direction
      previous_position = current_position
      return gerak baru
    ELSE
      return (0,0)  # tidak bergerak

END FUNCTION
```

```
  FUNCTION cari_tackle_target_in_radius(current_bot, board)
    IF ada targeted_enemy_id THEN
      cari bot dengan ID itu di board
      IF bot valid dan masih punya berlian cukup dan dalam radius THEN
        return bot
      ELSE
        targeted_enemy_id = None

    cari bot lain dalam radius yang berlian > threshold
    return bot terbaik jika ada, else None

  END FUNCTION

END
```

2. **Penjelasan Alur Program**

Pergerakan bot diatur oleh fungsi next_move dalam kode. Alur dimulai dengan inisialisasi awal bot, lalu bot memasuki fase penentuan tujuan utama dimulai dengan prioritas tertinggi. Prioritas pertama yaitu kembali ke base apabila inventory sudah penuh dan menyimpan poin diamond yang sudah dikumpulkan. Apabila inventory belum penuh, namun waktu sudah mau habis dan diamond di inventory lebih dari 0, maka bot pun akan kembali ke base. Apabila waktu belum habis, maka bot akan keliling mencari diamond.

Cara bot memilih diamond yang akan diambil duluan adalah dengan menghitung value dari setiap diamond yang ada dalam suatu radius tertentu dengan membagi poin diamond dengan jarak yang perlu ditempuh untuk mendapatkannya. Poin diamond diset 1 dan poin diamond merah diset 2 maka jika ada diamond dan diamond merah yang memiliki jarak yang sama, bot akan memprioritaskan diamond merah terlebih dahulu.

4.2 Struktur Data yang Digunakan

Struktur data yang digunakan dalam kode ini adalah :

1. Tuple (x, y)

   Digunakan untuk merepresentasikan posisi di papan permainan.

```python
self.previous_position = (None, None)
moves = [(0, 1), (0, -1), (1, 0), (-1, 0)]
```

2. GameObject

   GameObject adalah objek yang disediakan oleh *framework* game tempat bot ini berjalan.

   Mereka bertindak sebagai kontainer terstruktur untuk data terkait entitas game.

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

3. Variabel Skalar

   Bentuk angka tunggal atau nilai boolean.

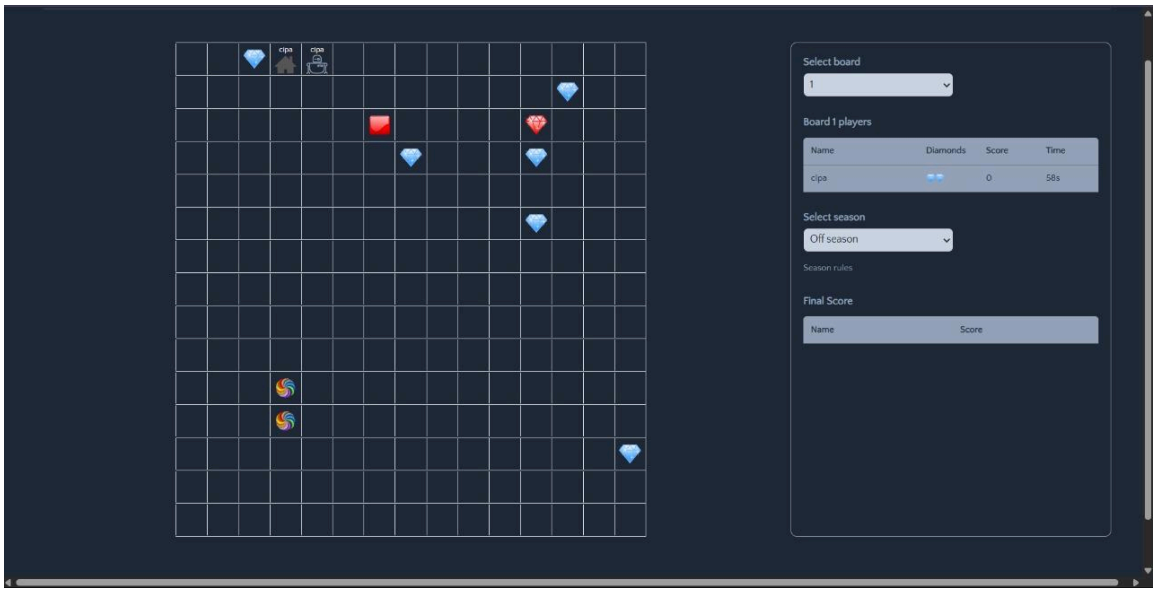
```python
self.red_diamond_score_multiplier = 2.0 # float
self.scan_radius = 5 # int
```

4.3 Pengujian Program

1. Skenario Pengujian

Pengujian Solo Run

Kondisi Awal Map

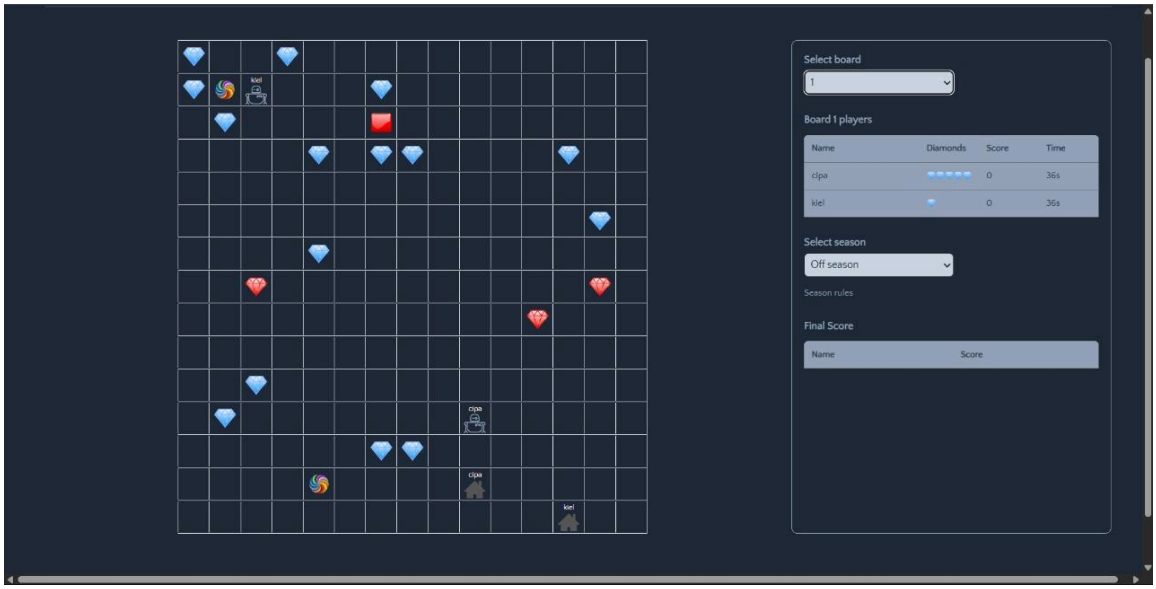


Peta Saat Bot Dijalankan

Final Score Bot

Pengujian Dengan Dua Bot

Bentuk Awal Map



Bentuk Saat 2 Bot Dijalankan

Final Score Bot

2. Hasil Pengujian dan Analisis

Pengujian Solo Run

Dalam pengujian ini, bot diaktifkan solo di papan permainan. Hasilnya menunjukkan bahwa bot ini efektif dalam mengumpulkan diamond dalam waktu singkat. Saat game dimulai, bot segera mengumpulkan diamond dengan jarak terdekat dari base dan saat inventory penuh, bot kembali ke base. Lalu bot berangkat lagi mengumpulkan diamond dengan jarak menengah dan kembali lagi saat inventory penuh. Untuk putaran terakhir, bot mengambil diamond dengan jarak jauh dan berhasil kembali ke base sebelum waktu habis.

Pengujian Dua Bot

Dalam pengujian ini, ada 2 bot yang diaktifkan di papan permainan. Hasilnya menunjukkan bahwa bot tidak se-efektif dalam mengumpulkan diamond dibanding saat ia aktif solo. Hal ini dapat disebabkan karena bot juga mempertimbangkan bot lawan dan mengambil langkah untuk menghindarinya dibanding mengumpulkan diamond. Bot hanya dapat mengumpulkan diamond dengan jarak dekat dari base dan kembali ke base. Lalu kembali mengambil 2 diamond terakhir dan kembali ke base.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan dari tugas besar ini adalah kode bot ini adalah contoh implementasi algoritma Greedy untuk bot dalam lingkungan game sederhana. Intinya, bot ini dirancang untuk selalu membuat keputusan yang tampak paling menguntungkan di saat itu, berdasarkan serangkaian prioritas yang telah ditentukan.

Bot ini diprogram untuk dapat membuat keputusan berdasarkan prioritas yang sudah ditentukan, seperti kembali ke base disaat inventory penuh, kembali ke base saat waktu sudah mau habis, dan mengumpulkan diamond dengan efektif. Jika bot stuck pun, ia akan melakukan gerakan random agar tidak kembali stuck. Bot ini dirancang untuk menghindari kemungkinan ia gagal menemukan langkah ke diamond atau kembali ke base.

5.2 Saran

Saran untuk pengembangan bot ini seperti :

1. Menambahkan kode untuk mereset map, untuk kode sekarang jika diamond sudah habis di map, ia akan keliling secara random hingga waktu habis.
2. Menambahkan kode untuk memprediksi lawan agar bot tidak memilih tujuan yang sama dengan bot lawan.

LAMPIRAN

A. Repository Github (link)

DAFTAR PUSTAKA

Modul 3 Perkuliahan Strategi Algoritma Institut Teknologi Sumatera *Algoritma Greedy (Bagian 1)*

Modul 3 Perkuliahan Strategi Algoritma Institut Teknologi Sumatera *Algoritma Greedy (Bagian 2)*