LAPORAN TUGAS BESAR

Strategi Algoritma
Semester Genap 2024/2025
Kelas RB

Oleh:

1. **Muhammad Dzaky** **123140039**
2. **Muhammad Farhan Muzakhi** **123140075**
3. **Bonifasius Ezra Mariano** **123140196**

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
LAMPUNG SELATAN

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 Setiap bot memiliki inventory dengan kapasitas terbatas dan harus kembali ke base untuk menyimpan diamond yang telah dikumpulkan agar score bertambah.

Permainan Diamonds ini terdiri dari beberapa komponen utama yaitu:

1. Diamonds: Diamonds sendiri terdapat dua jenis yaitu diamonds biru yang memiliki nilai 1 poin dan diamonds merah yang memiliki nilai 2 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 yang mengacak. posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.
3. Teleports: Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.
4. bots and bases: kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory.
5. Inventory: Bot memiliki inventory yang berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini memiliki kapasitas maksimum sehingga, sewaktu waktu bisa penuh. Agar inventory ini tidak penuh, bot bisa menyimpan isi inventory ke *base* agar inventory bisa kosong kembali.
6. Tackle System: Bot melakukan tackle terhadap bot lain. Bot yang di-tackle akan kehilangan semua diamond di inventory dan dikirim kembali ke base.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy adalah salah satu teknik dalam pemrograman yang digunakan untuk menyelesaikan masalah dengan cara membuat pilihan yang tampak optimal pada setiap langkah. Prinsip utama dari algoritma ini adalah memilih solusi terbaik yang tersedia saat ini, tanpa mempertimbangkan dampaknya pada langkah-langkah selanjutnya. algoritma greedy mempunyai beberapa langkah utama yaitu:

Inisialisasi yaitu Tahap awal untuk menyiapkan struktur data dan kondisi awal yang diperlukan sebelum algoritma greedy dijalankan, seperti mengatur variabel, mengurutkan elemen, atau mendefinisikan himpunan solusi kosong.

Seleksi**,**Langkah ini bertujuan memilih elemen terbaik berdasarkan kriteria tertentu yang dapat memberikan kontribusi maksimum terhadap solusi secara lokal.
Feasibility, Memastikan bahwa elemen terpilih tidak melanggar batasan masalah sehingga dapat dimasukkan ke dalam solusi secara valid.

Konstruksi Solusi Membangun solusi secara bertahap dengan menambahkan elemen-elemen yang telah lolos tahap seleksi dan feasibility. Iterasi Mengulangi proses seleksi hingga konstruksi solusi sampai seluruh elemen telah diproses atau solusi optimal tercapai.

Beberapa contoh permasalahan yang dapat diselesaikan secara optimal menggunakan algoritma greedy antara lain adalah masalah pencarian pohon rentang minimum (Minimum Spanning Tree) dengan algoritma Kruskal dan Prim. Meskipun algoritma greedy tergolong efisien dalam hal waktu dan memori, pendekatan ini tidak selalu menjamin solusi optimal untuk semua jenis masalah. Oleh karena itu, sebelum menerapkan algoritma greedy, penting untuk memastikan bahwa permasalahan yang dihadapi memenuhi kriteria yang sesuai.

2.1.1 Cara implementasi program

Program ini bekerja sebagai sistem permainan berbasis bot di mana setiap bot dikendalikan oleh logika tertentu yang dapat dikustomisasi. Dalam konteks ini, algoritma greedy diimplementasikan ke dalam logika bot untuk menentukan langkah terbaik berdasarkan kondisi terkini di papan permainan (game board). Bot akan bergerak dalam grid dua dimensi, dengan tujuan utama seperti mengumpulkan diamond sebanyak mungkin atau menyelesaikan objektif tertentu dengan efisien. Program terdiri atas dua komponen utama: game engine yang mensimulasikan dunia permainan dan bot sebagai agen cerdas yang berinteraksi dalam lingkungan tersebut.
Implementasi program menggunakan arsitektur client-server, dimana game engine bertindak sebagai server yang mengatur seluruh jalanya permainan, dan ada bot sebagai client yang

melakukan aksi berdasarkan strategi yang telah diprogram, dengan membuat kelas baru didalam direktori game logic untuk mengatur bot, mewarisi inherit dari kelas baselogic, mengimplementasikan method dengan next_move yang akan dipanggil secara berkala oleh sistem, nilai yang dikembalikan berupa arah gerakan delta_x, delta_y seperti 0,1/ 1,0, logika ini akan memilih langkah yang memberikan keuntungan maksimal secara langsung pada saat itu, seperti mendekati diamond yang paling dekat terlebih dahulu

Menjalankan Game Engine

1. Cara menjalankan *game engine*

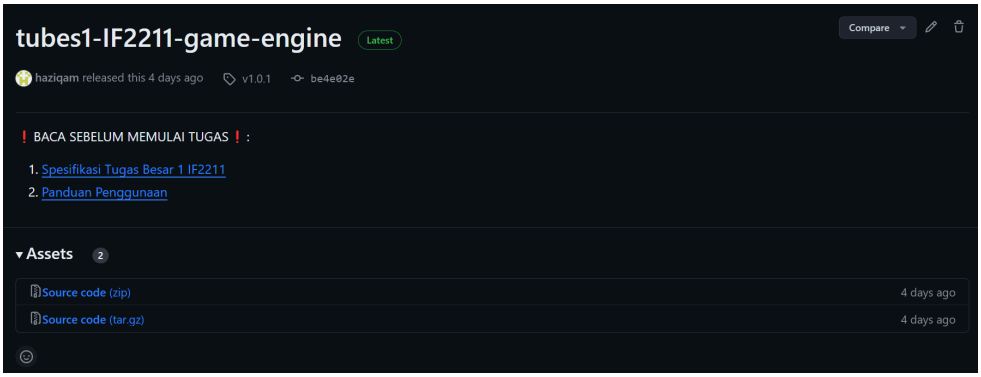
a. *Requirement* yang harus di-install

- Node.js (https://nodejs.org/en)
- Dockerdesktop (https://www.docker.com/products/docker-desktop/)
- Yarn

```
npm install --global yarn
```

b. Instalasi dan konfigurasi awal

1) Download source code (.zip) pada release game engine



2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal

3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-game-engine-1.1.0
```

4) Install dependencies menggunakan Yarn

```
yarn
```

5) Setup default environment variable dengan menjalankan script berikut

Untuk Windows

```
./scripts/copy-env.bat
```

` Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/copy-env.sh
./scripts/copy-env.sh
```

6) Setup local database (buka aplikasi docker desktop terlebih dahulu, lalu jalankan command berikut di terminal)

```
docker compose up -d database
```

Lalu jalankan script berikut. Untuk Windows

```
./scripts/setup-db-prisma.bat
```

Untuk Linux / (possibly) macOS

```
chmod +x ./scripts/setup-db-prisma.sh
./scripts/setup-db-prisma.sh
```

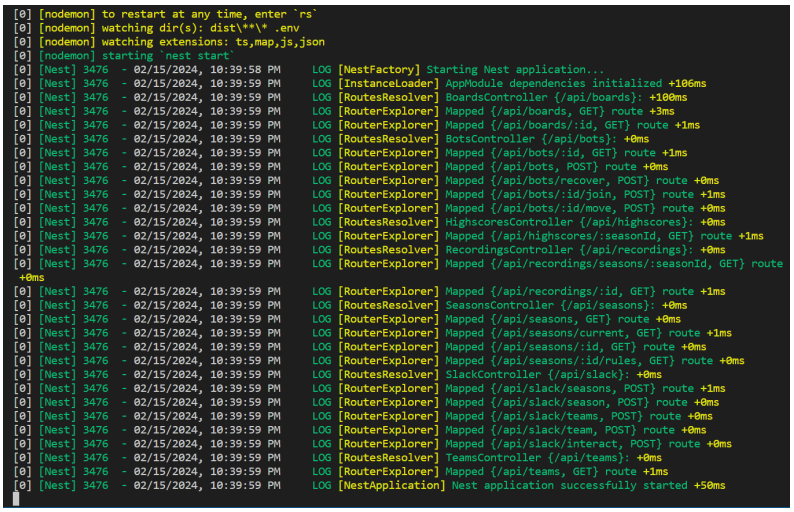
c. *Build*

```
npm run build
```

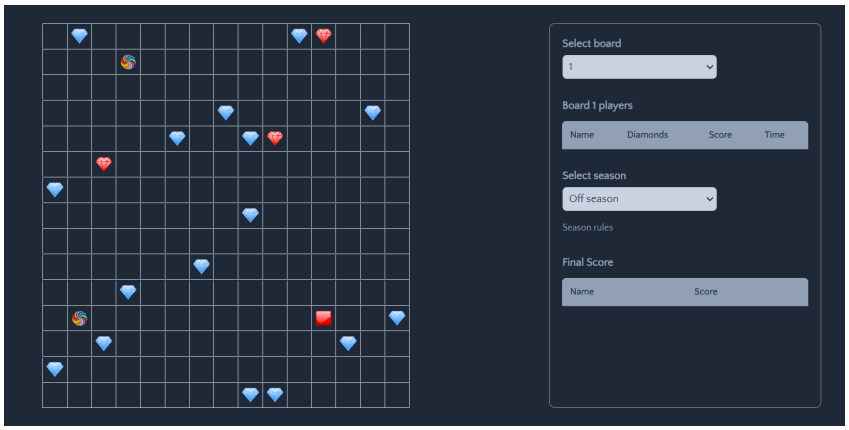
d. *Run*

```
npm run start
```

Jika berhasil, tampilan terminal akan terlihat seperti gambar di bawah ini.



Kunjungi *frontend* melalui http://localhost:8082/. Berikut adalah tampilan awal *frontend*.



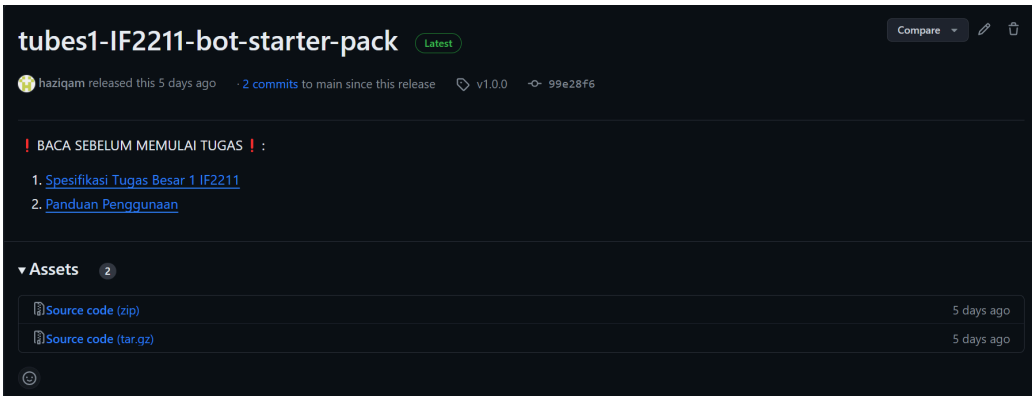
2.1.2 Menjalankan bot program

a. *Requirement* yang harus di-install

- Python (https://www.python.org/downloads/)

b. Instalasi dan konfigurasi awal

1) Download source code (.zip) pada release bot starter pack



2) Extract zip tersebut, lalu masuk ke folder hasil extractnya dan buka terminal
3) Masuk ke root directory dari project (sesuaikan dengan nama rilis terbaru)

```
cd tubes1-IF2110-bot-starter-pack-1.0.1
```

4) Install dependencies menggunakan pip

```
pip install -r requirements.txt
```

c. Run

Untuk menjalankan satu bot (pada contoh ini, kita menjalankan satu bot dengan logic yang terdapat pada file game/logic/random.py)

```
python main.py --logic Random --email=your_email@example.com --name=your_name --password=your_password --team etimo
```

Untuk menjalankan beberapa bot sekaligus (pada contoh ini, kita menjalankan 4 bot dengan logic yang sama, yaitu game/logic/random.py)

- Untuk windows

```
./run-bots.bat
```

- Untuk Linux / (possibly) macOS

```
./run-bots.sh
```

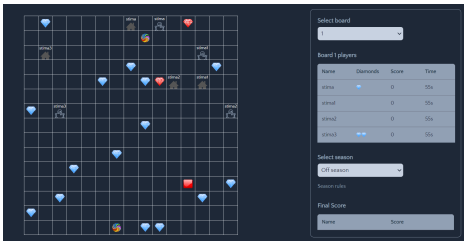
Kalian dapat menyesuaikan *script* yang ada pada run-bots.bat atau run-bots.sh dari segi **logic yang digunakan, email, nama,** dan **password**

run-bots.bat

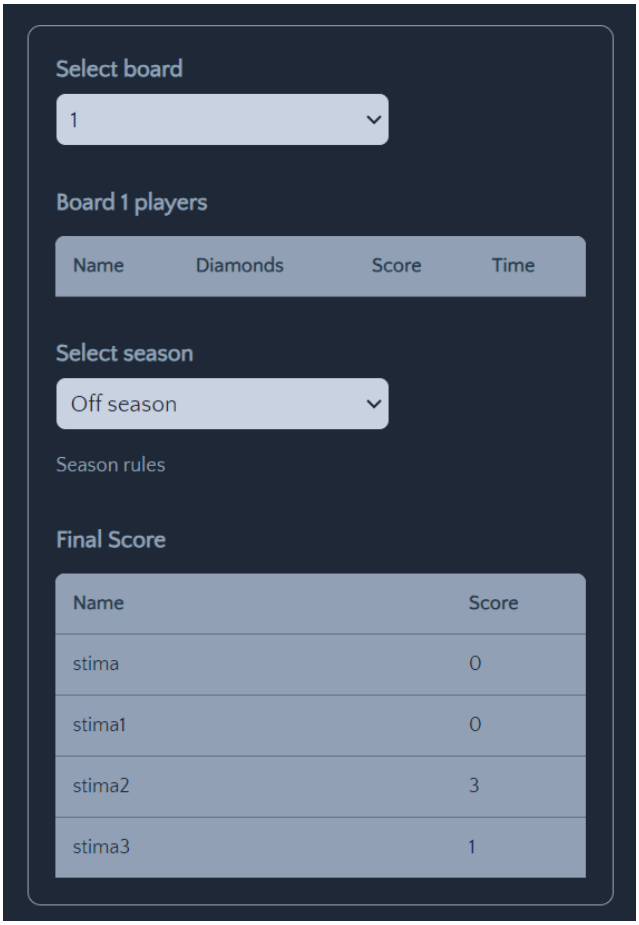
```
@echo off
start cmd /c "python main.py --logic Random --email=test@email.com --name=stima --password=123456 --team etimo"
start cmd /c "python main.py --logic Random --email=test1@email.com --name=stima1 --password=123456 --team etimo"
start cmd /c "python main.py --logic Random --email=test2@email.com --name=stima2 --password=123456 --team etimo"
start cmd /c "python main.py --logic Random --email=test3@email.com --name=stima3 --password=123456 --team etimo"

```

Perhatikan bot yang telah bergabung melalui *frontend*



Saat permainan selesai, final score akan muncul pada sisi kanan bawah.



BAB III
APLIKASI STRATEGI GREEDY

3.1 Proses *Mapping*

Dalam pembuatan algoritma bot permainan Diamonds, algoritma greedy digunakan sebagai strategi pengambilan keputusan. Strategi ini dipilih karena sifatnya yang efisien secara komputasi dan cocok untuk lingkungan permainan dinamis dengan keterbatasan waktu dan informasi. Permasalahan dalam permainan Diamonds dimodelkan ke dalam elemen-elemen dasar algoritma greedy sebagai berikut:

1) **Himpunan kandidat** berisikan seluruh objek yang ada di permainan, seperti diamond, tombol merah, teleport, dan base.
2) **Himpunan solusi** yaitu langkah -langkah yang diambil oleh bot untuk mencapai tujuan tertentu seperti mengambil diamond.
3) **Fungsi solusi** yaitu pengumpulan diamond sebanyak dan seefisien mungkin.
4) **Fungsi seleksi** yaitu terjadi seleksi jarak antara diamond biru atau merah, sebelum bot bejalan.
5) **Fungsi kelayakan** yaiut pengecekan inventory jika penuh tidak akan mengambil diamond.
6) **Fungsi objektif** yaitu terus mencari diamond sampai inventory penuh dan ke base, kemudian terus berulang.

3.2 Strategi Greedy

Algoritma yang dibuat perlu adanya strategi eksplorasi untuk mencapai tujuan antara lain, memilih diamond yang paling dekat dengan bot, memilih diamond dengan nilai tertinggi, dan memilih diamond dengan perbandingan nilai terhadap jarak yang paling jauh. Setiap strategi memiliki kelebihan dan kekurangan. Strategi berdasarkan jarak saja lebih bagus dalam kecepatan namun tidak cocok untuk score. Strategi berdasarkan nilai |akan memakan waktu yang lebih lama. Sementara itu, strategi berdasarkan perbandingan menghasilkan memungkinkan performa terbaik, karena memungkinkan menyeimbangkan antara waktu tempuh dan nilai. sehingga strategi yang baik yaitu pada perbandingan antara jarak dan nilai, karena dinilai akan memberikan hasil yang baik dari segi efisiensi gerakan maupun jumlah nilai yang dikumpulkan

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi algoritma greedy

4.1.1 Pseudocode

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


class HcBot(BaseLogic):
    def __init__(self):
        self.goal_position = None
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]  # Arah dasar: Timur, Selatan, Barat, Utara (untuk roaming fallback)
        self.current_direction_index = 0

    def manhattan_distance(self, pos1: Position, pos2: Position) -> int:
        """Menghitung jarak Manhattan antara dua posisi."""
        return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

    def get_diamonds_in_radius(self, center_pos: Position, radius: int, diamonds_on_board: list[GameObject]) -> list[GameObject]:
        """Mengembalikan daftar diamond dalam radius Manhattan tertentu dari posisi bot."""
        diamonds_in_range = []
        for diamond in diamonds_on_board:
            if self._manhattan_distance(center_pos, diamond.position) <= radius:
                diamonds_in_range.append(diamond)
        return diamonds_in_range

    def update_goal_position(self, board_bot: GameObject, board: Board):
        """
        Menentukan posisi tujuan (goal_position) bot berdasarkan prioritas.
        Strategi Greedy digunakan untuk memilih opsi terbaik pada saat itu:
        1. Pulang ke base jika inventaris penuh
        2. Menekan tombol merah jika tidak ada diamond di radius tertentu
```

```
        3. Ambil diamond dengan nilai terbaik per jarak
        4. Roaming (fallback jika tidak ada target)
        """
        current_position = board_bot.position
         inventory_size = board_bot.properties.inventory_size if board_bot.properties.inventory_size
is not None else 5

        # Kumpulan objek di papan permainan
                    diamonds_on_board = [obj for obj in board.game_objects if obj.type ==
"DiamondGameObject"]
                red_button_on_board = next((obj for obj in board.game_objects if obj.type ==
"RedButtonGameObject"), None)

        # Reset tujuan dan skor terbaik
        self.goal_position = None
        best_overall_score = -1.0

        # (1) Strategi Greedy Prioritas Tinggi: Pulang ke base jika inventaris penuh
        if board_bot.properties.diamonds >= inventory_size:
            base_position = board_bot.properties.base
            self.goal_position = base_position

            # Jika sudah di base, tidak perlu gerak lagi
            if position_equals(current_position, self.goal_position):
                return

        # (2) Strategi Heuristik Greedy: Evaluasi tombol merah atau diamond terbaik
        if self.goal_position is None:
            DIAMOND_CHECK_RADIUS_FOR_BUTTON = 15  # Radius untuk mengecek apakah
ada diamond di sekitar

                        diamonds_in_current_radius = self._get_diamonds_in_radius(current_position,
DIAMOND_CHECK_RADIUS_FOR_BUTTON, diamonds_on_board)

            # (2a) Tombol Merah diprioritaskan jika tidak ada diamond di radius (Greedy Conditional)
            if red_button_on_board and not diamonds_in_current_radius:
                self.goal_position = red_button_on_board.position
                best_overall_score = float('inf')  # Skor tertinggi untuk memastikan tombol dipilih
```

```
                print(f'Menggunakan Tombol Merah: Tidak ada diamond dalam radius {DIAMOND_CHECK_RADIUS_FOR_BUTTON}.')
            return

        # (2b) Evaluasi semua diamond berdasarkan nilai / jarak (Greedy Heuristic)
        for diamond in diamonds_on_board:
                diamond_value = diamond.properties.points if diamond.properties and diamond.properties.points is not None else 1

            # Prioritaskan diamond bernilai 1 jika hanya satu slot tersisa
            if board_bot.properties.diamonds == (inventory_size - 1) and diamond_value != 1:
                continue

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

            if distance == 0:
                current_score = float('inf')  # Sudah berada di atas diamond
            else:
                # Skor berbasis rasio nilai terhadap jarak (Greedy Value/Cost Ratio)
                current_score = diamond_value / distance

            # Simpan diamond dengan skor terbaik
            if current_score > best_overall_score:
                best_overall_score = current_score
                self.goal_position = diamond.position

    # Jika tidak ada tujuan ditemukan, self.goal_position akan tetap None => mode roaming

  def next_move(self, board_bot: GameObject, board: Board):
    """
    Menghitung langkah bot selanjutnya.
    Menggunakan hasil goal_position dari _update_goal_position dan memilih arah terbaik.
    """
    current_position = board_bot.position

    # Perbarui posisi tujuan berdasarkan strategi greedy
    self._update_goal_position(board_bot, board)
```

```
        delta_x, delta_y = 0, 0

        # (3) Pergerakan menuju goal_position (Greedy Direct Move)
        if self.goal_position:
            # Jika sudah sampai tujuan, reset agar evaluasi ulang di giliran berikutnya
            if position_equals(current_position, self.goal_position):
                # Berhenti jika sudah di base dan inventaris penuh
                        if board_bot.properties.diamonds >= (board_bot.properties.inventory_size if
board_bot.properties.inventory_size is not None else 5) and position_equals(current_position,
board_bot.properties.base):
                    return (0, 0)
                self.goal_position = None
                return (0, 0)

            # Hitung arah ke goal
            delta_x, delta_y = get_direction(
                current_position.x,
                current_position.y,
                self.goal_position.x,
                self.goal_position.y,
            )

            # (4) Fallback Greedy: Validasi langkah, coba alternatif jika terhalang
            if not board.is_valid_move(current_position, delta_x, delta_y):
                # Coba bergerak hanya ke arah X atau Y
                if delta_x != 0 and board.is_valid_move(current_position, delta_x, 0):
                    delta_y = 0
                elif delta_y != 0 and board.is_valid_move(current_position, 0, delta_y):
                    delta_x = 0
                else:
                    # Roaming fallback: coba arah berdasarkan rotasi default
                    temp_delta_x, temp_delta_y = self.directions[self.current_direction_index]
                    if board.is_valid_move(current_position, temp_delta_x, temp_delta_y):
                        delta_x, delta_y = temp_delta_x, temp_delta_y
                                    self.current_direction_index = (self.current_direction_index + 1) %
len(self.directions)
                    else:
                        return (0, 0)  # Semua arah buntu
```

```
        # (5) Roaming fallback jika tidak ada goal_position (Greedy Random Roaming)
        if not self.goal_position:
            temp_delta_x, temp_delta_y = self.directions[self.current_direction_index]
            if board.is_valid_move(current_position, temp_delta_x, temp_delta_y):
                delta_x, delta_y = temp_delta_x, temp_delta_y
                self.current_direction_index = (self.current_direction_index + 1) % len(self.directions)
            else:
                # Coba arah lain secara rotasi jika arah saat ini buntu
                for _ in range(len(self.directions)):
                    self.current_direction_index = (self.current_direction_index + 1) % len(self.directions)
                    temp_delta_x, temp_delta_y = self.directions[self.current_direction_index]
                    if board.is_valid_move(current_position, temp_delta_x, temp_delta_y):
                        delta_x, delta_y = temp_delta_x, temp_delta_y
                        break
                else:
                    return (0, 0)  # Semua arah buntu

        return delta_x, delta_y
```

4.1.2 Penjelasan alur program

Program ini terdiri dari class HcBot yang mewarisi dari BaseLogic, dan memiliki dua fungsi utama:

a. update_goal_position

Fungsi ini memilih **tujuan (goal_position)** berdasarkan kondisi dan strategi:

1. **Pulang ke base** jika diamond yang dibawa mencapai kapasitas maksimal.
2. **Tekan tombol merah** jika tidak ada diamond dalam radius tertentu.
3. **Pilih diamond terbaik** berdasarkan rasio nilai / jarak (*greedy ratio*).
4. **Fallback roaming**, jika tidak ada tujuan ditemukan.

b. next move

Fungsi ini menentukan langkah selanjutnya berdasarkan goal_position:

1. Jika sudah di goal_position, berhenti.
2. Jika belum, hitung arah menuju tujuan dengan get_direction().
3. Validasi apakah gerakan tersebut legal. Jika tidak:Coba hanya di sumbu X atau Y.Gunakan fallback roaming: coba arah (timur, selatan, barat, utara) secara bergiliran.
4. Jika tidak ada goal_position, tetap roaming.

4.2 Struktur data yang digunakan

Struktur	Deskripsi
GameObject	Objek dalam game seperti bot, diamond, tombol merah, base. Memiliki atribut seperti position, type, dan properties
Position	Posisi dalam bentuk koordinat (x, y).
Board	Memiliki fungsi seperti is_valid_move() untuk mengecek langkah valid.
list[GameObject]	Digunakan untuk menyimpan semua objek di papan, seperti daftar diamond.
tuple	Digunakan untuk menyimpan arah gerakan (delta_x, delta_y) serta daftar arah dasar [(1,0), (0,1),(-1,0), (0,-1)].

4.3 Pengujian program

4.3.1 Skenario Pengujian

Skenario 1: Jika bot membawa diamond hingga penuh makan Bot harus pulang ke base.

Skenario 2: Jika tidak ada diamond dalam radius 15 maka Bot harus menuju tombol merah.

Skenario 3: Jika banyak diamond tersebar di papan maka Bot harus menuju diamond terbaik (nilai tinggi / jarak dekat).

Skenario 4: Jika bot sudah berada di posisi tujuan maka Bot tidak perlu bergerak.

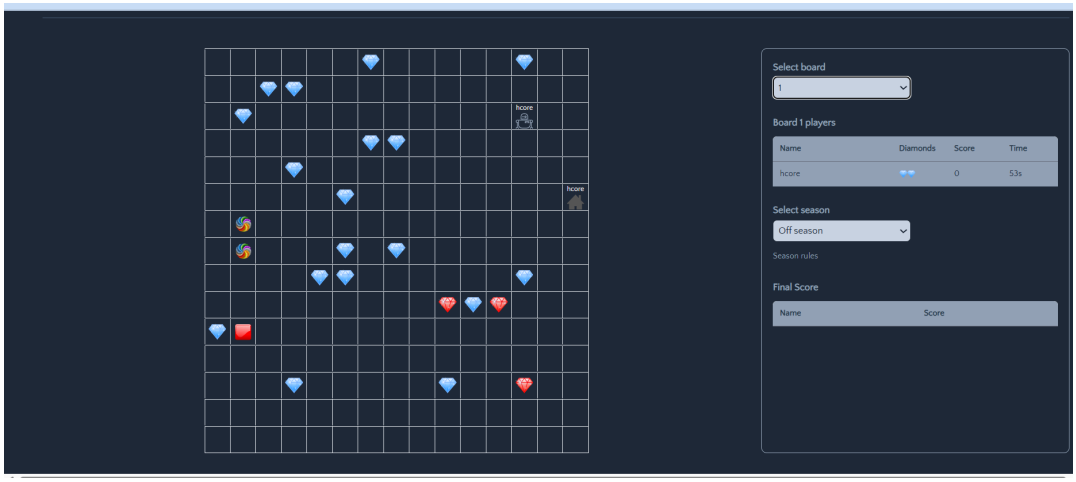
Skenario 5: Jika semua arah terhalang (tidak valid) maka Bot tetap diam (tidak

bergerak).

Skenario 6: Jika tidak ada tujuan dan semua arah valid maka Bot roaming mengikuti arah rotasi dasar.

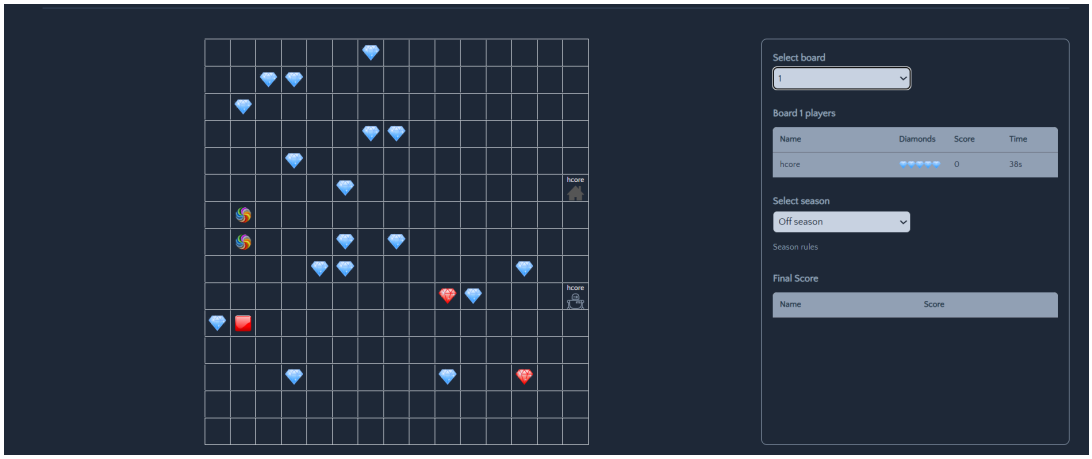
4.3.2 Hasil Pengujian dan analisis

1) **Pengujian 1**:



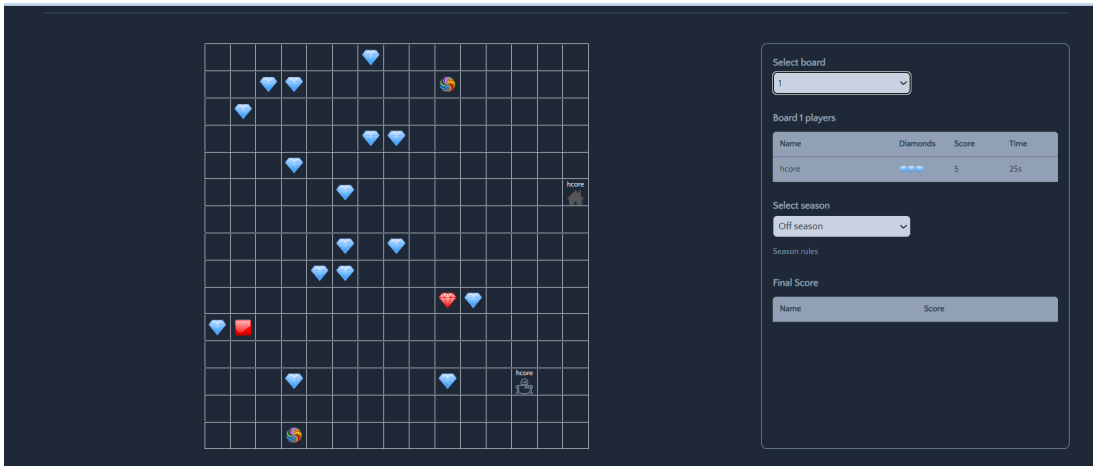
Bot akan mencari diamond di sekitar .

2) **Pengujian 2**: .

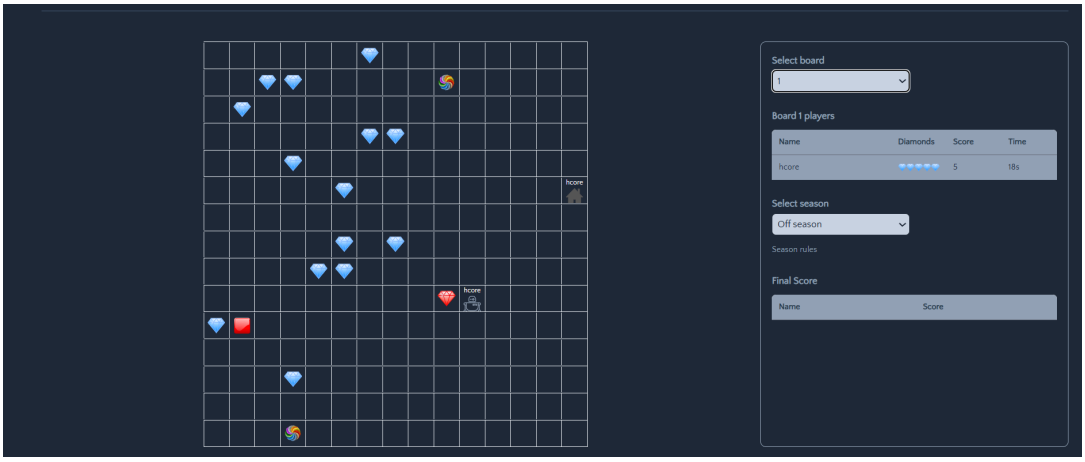


Diketahui Inventaris penuh, maka Bot langsung menuju base.Hasil: sesuai ekspektasi

3) **Pengujian 3**:



bot mengambil di titik 10, 13, kemudian Terdapat dua diamond, bot mengambil diamond biru karena inventori hanya cukup 1.



BAB V
KESIMPULAN

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian terhadap strategi algoritma greedy dalam permainan *Diamonds*, dapat disimpulkan bahwa pendekatan ini mampu memberikan performa yang cukup efisien dalam pengambilan keputusan secara real-time. Strategi greedy bekerja dengan memilih langkah yang secara lokal memberikan keuntungan maksimal, seperti mendekati diamond dengan rasio nilai terhadap jarak tertinggi atau kembali ke base saat inventory penuh.

Pemanfaatan strategi ini terbukti efektif untuk menyelesaikan skenario pada permainan ini, termasuk kondisi ketika diamond tersebar tidak merata, tombol merah perlu diaktifkan, atau saat bot menghadapi hambatan gerak. Bot mampu beradaptasi dengan perubahan kondisi papan permainan melalui logika prioritas yang terstruktur dengan baik, seperti pemilihan tujuan, validasi gerakan, hingga mekanisme fallback ketika tidak ada tujuan yang jelas.

Secara umum, algoritma greedy yang diterapkan telah memenuhi ekspektasi dalam hal efisiensi pergerakan dan pencapaian skor. Meskipun tidak menjamin solusi global yang optimal, strategi ini memberikan hasil yang kompetitif dalam ruang lingkup permainan dengan kompleksitas dinamis dan keterbatasan waktu proses.

5.2 Saran

Perbaikan Evaluasi Strategi
Menambahkan analisis prediktif agar bot dapat memperkirakan situasi papan ke depan, tidak hanya fokus pada kondisi saat ini.
Kombinasi dengan Algoritma Lain
Menggabungkan greedy dengan algoritma lain, untuk hasil yang lebih optimal dalam kondisi kompleks.
Manajemen Risiko
Menambahkan logika untuk menghindari tackle dan memanfaatkan teleporter secara lebih strategis.
Optimasi Performa
Melakukan uji coba pada skenario lebih kompleks untuk menyempurnakan efisiensi waktu dan penggunaan memori.

LAMPIRAN

Link Sourc code : https://github.com/14-039-MuhammadDzaky/Tubes1_HardCore

DAFTAR PUSTAKA

(*Perancangan Aplikasi Game Maze Escape Dengan Menerapkan Algoritma Greedy Untuk Pencarian Jalur Terpendek*, 2021)