IMPLEMENTASI ALGORITMA GREEDY PADA PENGEMBANGAN BOT CERDAS DALAM PERMAINAN *DIAMONDS*

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 6 - DuhBot

Aditya Ronal Maruli	123140093
Nabila Yuliana	123140099

Dosen Pengampu: Imam Eko Wicaksono, S.Si., M.Si.

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

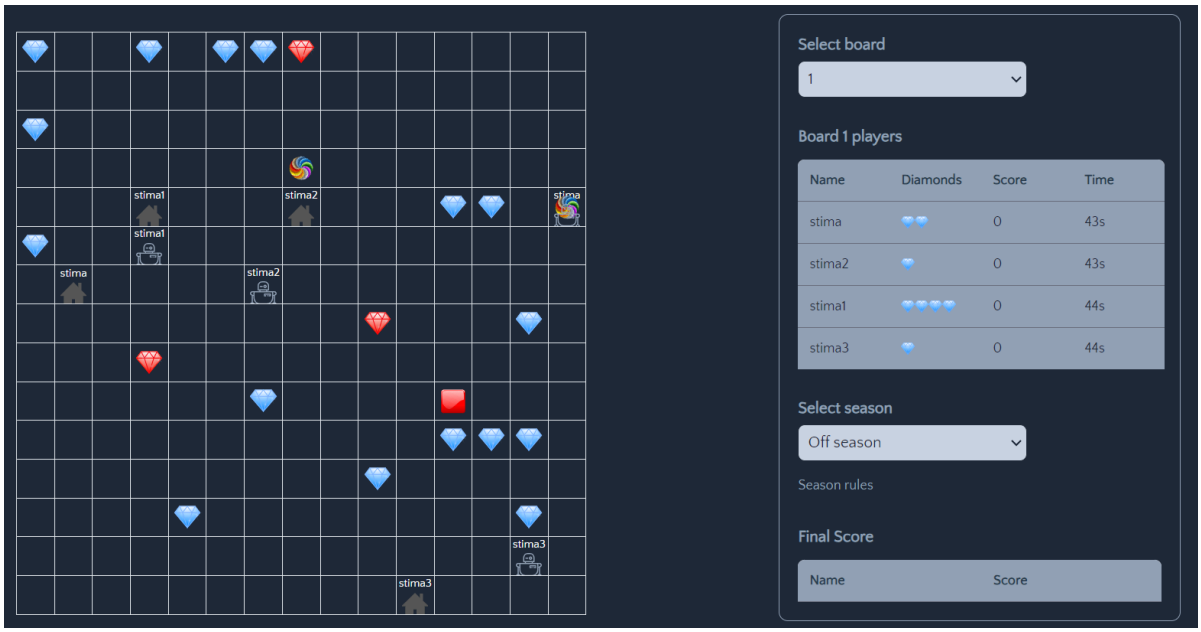
INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Diamonds merupakan sebuah tantangan pemrograman yang mengharuskan para peserta mengembangkan dan mengadu bot ciptaan mereka satu sama lain. Masing-masing peserta memiliki bot yang ditugaskan untuk mengumpulkan sebanyak mungkin *diamond*. Tantangan ini tidaklah mudah, karena berbagai rintangan disiapkan untuk menambah tingkat kesulitan dan keseruan permainan. Untuk keluar sebagai pemenang, peserta perlu merancang strategi khusus yang diimplementasikan pada bot mereka. Penjelasan lebih rinci mengenai aturan permainan akan disampaikan di bagian selanjutnya.



Gambar 1.1 Layar Permainan Diamonds

Pada tugas ini, mahasiswa diminta untuk merancang sebuah bot yang akan dipertandingkan dengan bot milik mahasiswa lainnya. Dalam proses pengembangan bot tersebut, mahasiswa diwajibkan menerapkan ***strategi greedy*** sebagai pendekatan utama.

Program permainan *Diamonds* terdiri atas :

1. *Game engine*, yang secara umum berisi :
    a. Kode *backend* permainan, yang berisi *logic* permainan serta API yang disediakan untuk berkomunikasi *frontend* dan program bot
    b. Kode *frontend* permainan, yang berfungsi untuk memvisualisasikan permainan
2. *Bot starter pack*, yang secara umum berisi :
    a. Program untuk memanggil API yang tersedia pada *backend*
    b. Program *bot logic* (bagian ini yang akan diimplementasikan dengan algoritma *greedy* untuk bot kelompok)
    c. Program utama (*main*) dan utilitas lainnya

Komponen-komponen dari permainan *Diamonds* antara lain :

1. **Diamonds**

    Untuk memenangkan pertandingan, kita harus mengumpulkan *diamond* sebanyak-banyaknya dengan melewati atau 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*.

Gambar 1.2 Varian Diamond Biru

Gambar 1.3 Varian Diamond Merah

2. **Red Button / Diamond Button**

   Ketika *red button* ini dilewati atau 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.

Gambar 1.4 Red Button / Diamond Button

3. **Teleporters**

   Terdapat 2 *teleporter* yang saling terhubung satu sama lain. Jika bot melewati sebuah *teleporter* maka bot akan berpindah menuju posisi *teleporter* yang lain.

Gambar 1.5 Teleporters

4. **Bots**

   Pada game ini kita akan menggerakkan bot untuk mendapatkan *diamond* sebanyak-banyaknya. Bot ini bisa men-*tackle* bot lainnya.

Gambar 1.6 Bots "Stima"

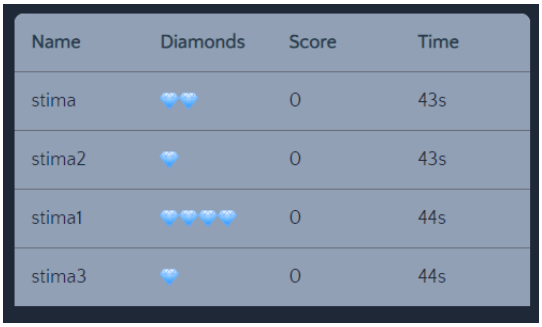
5. **Bases**

   Setiap bot dilengkapi dengan sebuah *Base*, yang berfungsi untuk menyimpan *diamond* yang sedang dibawa.

Gambar 1.7 Base "Stima"

6. **Inventory**

   Bot dilengkapi dengan sebuah inventaris, yang berperan sebagai lokasi penyimpanan sementara untuk *diamond* yang telah dikumpulkan. Inventaris ini dibatasi oleh sebuah kapasitas maksimal, yang artinya dapat terisi penuh. Untuk menghindari keadaan penuh tersebut, bot dapat mentransfer isi inventaris mereka ke *Base*, sehingga memungkinkan inventaris tersebut untuk dikosongkan kembali.



Gambar 1.8 Layar Inventory

Untuk mengetahui *flow* dari game ini, berikut ini adalah cara kerja permainan *Diamonds* :

1. Pertama, setiap pemain (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 diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil *diamond-diamond* yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, *diamond* yang berwarna merah memiliki 2 poin dan *diamond* yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah *inventory*, dimana *inventory* berfungsi sebagai tempat penyimpanan sementara *diamond* yang telah diambil. *Inventory* ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke *home base*.
5. Apabila bot menuju ke posisi *home base*, *score* bot akan bertambah senilai *diamond* yang tersimpan pada *inventory* dan *inventory* bot akan menjadi kosong kembali.

6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke *home base* dan semua *diamond* pada *inventory* bot B akan hilang, diambil masuk ke *inventory* bot A (istilahnya *tackle*).
7. Selain itu, terdapat beberapa fitur tambahan seperti *teleporter* dan *red button* yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. *Score* masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

Dalam tugas besar ini, mahasiswa diminta untuk bekerja dalam kelompok minimal dua orang dan maksimal tiga orang. Tujuan dari tugas ini adalah untuk mengimplementasikan algoritma *Greedy* terbaik yang berkaitan dengan fungsi objektif permainan, yaitu mengumpulkan *diamond* sebanyak mungkin dan mencegah *diamond* tersebut diambil oleh bot lain.

Setiap kelompok wajib menjelaskan secara eksplisit strategi *Greedy* yang digunakan dalam laporan, serta menyertakan kode program yang sesuai. Kode harus dibuat sendiri tanpa menyalin dari internet dan setiap kelompok diharapkan untuk membuat program sendiri dengan menggunakan kreativitas masing-masing. Program yang dibuat harus dapat dijalankan pada game *engine* yang telah ditentukan dan mampu bersaing dengan bot dari kelompok lain.

Strategi *Greedy* yang diimplementasikan harus disertai komentar yang jelas pada bagian kode program yang relevan. Kelompok yang membuat penjelasan strategi dna simulasi permainan bot, dengan menampilkan wajah semua anggotanya, akan mendapatkan bonus poin.

Asistensi tugas besar bersifat opsional, namun mahasiswa dapat meminta bimbingan dari asisten yang ditunjuk jika diperlukan. Bot dari setiap kelompok akan mengikuti kompetisi terbuka yang disaksikan seluruh peserta kuliah. Kelompok pemenang akan menerima hadiah menarik. Informasi terkait pendataan kelompok, asistensi, demo dan pengumpulan laporan dapat diakses melalui tautan yang telah disediakan.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah (*step by step*) sedemikian sehingga pada setiap langkah :

1. Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip *"take what you can get now!"*).
2. Dan "berharap" bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Dalam menjalankan ***Algoritma Greedy***, terdapat beberapa elemen penting yang mempermudah proses penyelesaian masalah, yaitu :

1. **Himpunan kandidat (C)** : berisi elemen yang dapat dipilih pada langkah, seperti simpul, sisi, koin atau task.
2. **Himpunan solusi (S)** : kumpulan kandidat yang telah dipilih sebagai bagian dari solusi.
3. **Fungsi solusi** : menentukan apakah solusi akhir telah tercapai.
4. **Fungsi seleksi** : memilih kandidat terbaik berdasarkan strategi greedy (heuristik).
5. **Fungsi kelayakan** : memeriksa apakah kandidat layak dimasukkan ke dalam solusi.
6. **Fungsi objektif** : digunakan untuk memaksimalkan atau meminimalkan hasil sesuai tujuan masalah.

Setelah seluruh elemen tersebut terpenuhi, proses pencarian solusi dapat dilakukan. ***Algoritma Greedy*** bekerja dengan mencari himpunan bagian (S) dari himpunan kandidat (C), di mana S harus memenuhi kriteria tertentu, yaitu membentuk suatu solusi yang sah dan dioptimalkan berdasarkan fungsi objektif. Algoritma ini menghasilkan sejumlah solusi optimum lokal, kemudian memilih yang terbaik diantara solusi-solusi tersebut sebagai solusi optimum global.[1]

Namun, solusi optimum global yang diperoleh dari ***Algoritma Greedy*** belum tentu merupakan solusi terbaik secara keseluruhan, karena bisa saja hanya berupa solusi *sub-optimum* atau semu. Hal ini disebabkan oleh dua faktor utama, yaitu :

1. ***Algoritma Greedy*** tidak mengevaluasi seluruh kemungkinan solusi secara menyeluruh seperti pada metode pencarian menyeluruh (*exhaustive search*).
2. Tersedia berbagai pilihan fungsi seleksi, sehingga pemilihan fungsi yang tepat menjadi penting agar solusi yang dihasilkan benar-benar optimal.

Sebagai kesimpulan, ***Algoritma Greedy*** tidak selalu mampu memberikan solusi terbaik untuk semua jenis permasalahan. Namun, jika solusi mutlak tidak menjadi keharusan, algoritma ini dapat dimanfaatkan untuk menghasilkan solusi hampiran (*approximation*), terutama bila dibandingkan dengan algoritma lain yang memerlukan waktu komputasi eksponensial. Misalnya, dalam permasalahan pencarian lintasan dengan bobot minimum pada *Traveling Salesman Problem* (TSP) dengan banyak simpul, *algoritma brute force* akan memerlukan waktu yang sangat lama. Sebaliknya, *algoritma greedy* mampu memberikan solusi lebih cepat meskipun belum tentu optimal, tetapi dalam menghasilkan solusi optimal sering dibuktikan dengan *counterexample* yang menunjukkan adanya solusi yang lebih baik.

Dalam penerapannya, terdapat cukup banyak jenis permasalahan yang dapat diselesaikan menggunakan pendekatan *greedy*, diantaranya adalah sebagai berikut :

1. Persoalan penukaran uang (*coin exchange problem*)
2. Persoalan memilih aktivitas (*activity selection problem*)
3. Minimasi waktu di dalam sistem
4. Persoalan *knapsack* (*knapsack problem*)
5. Penjadwalan *job* dengan tenggat waktu (*job scheduling with deadlines*)
6. Pohon merentang minimum (*minimum spanning tree*)
7. Lintasan terpendek (*shortest path*)
8. Kode Huffman (*huffman code*)
9. Pecahan Mesir (*egyptian fraction*)

2.2 Cara Kerja Program

Secara umum, program yang dibuat bertujuan untuk mengontrol bot dalam permainan *Diamonds*, agar dapat mengumpulkan *diamond* sebanyak mungkin menggunakan strategi *algoritma greedy*. Program ini bekerja dengan menganalisis kondisi *board* secara real-time dan membuat keputusan terbaik yang tersedia saat itu, tanpa mempertimbangkan kemungkinan langkah masa depan.

Bot akan terus berkomunikasi dengan *game engine* melalui API yang telah disediakan. Setiap langkah pergerakan bot ditentukan dengan mengevaluasi beberapa opsi berdasarkan nilai keuntungan lokal terbesar – misalnya, jarak terdekat ke *diamond* bernilai tinggi, kapasitas inventory dan risiko bertabrakan dengan bot lawan.

1. **Proses Pelaksanaan Aksi oleh Bot**

   Permainan dalam tugas besar ini menggunakan basis web, sehingga setiap tindakan yang dilakukan-–mulai dari proses pendaftaran bot hingga pelaksanaan aksi pergerakan bot—dilakukan melalui HTTP *request* ke *endpoint* API yang telah disediakan oleh *backend*. Berikut adalah urutan *request* yang terjadi dari awal mula permainan.

   a. Program bot akan mengecek apakah bot sudah terdaftar atau belum, dengan mengirimkan *POST request* terhadap *endpoint* /api/bots/recover dengan *body* berisi *email* dan *password* bot. Jika bot sudah terdaftar, maka *backend* akan memberikan *response code* 200 dengan *body* berisi id dari bot tersebut. Jika tidak, *backend* akan memberikan *response code* 404.
   b. Jika bot belum terdaftar, maka program bot akan mengirimkan *POST request* terhadap *endpoint* /api/bots dengan *body* berisi *email, name, password*, dan *team*. Jika berhasil, maka *backend* akan memberikan *response code* 200 dengan *body* berisi id dari bot tersebut.
   c. Ketika id bot sudah diketahui, bot dapat bergabung ke *board* dengan mengirimkan *POST request* terhadap *endpoint* /api/bots/{id}/join dengan *body* berisi board id yang diinginkan (preferredBoardId). Apabila bot berhasil bergabung, maka *backend* akan memberikan *response code* 200 dengan *body* berisi informasi dari *board*.
   d. Program bot akan mengkalkulasikan *move* selanjutnya secara berkala berdasarkan kondisi *board* yang diketahui, dan mengirimkan *POST request* terhadap *endpoint* /api/bots/{id}/move dengan *body* berisi *direction* yang akan ditempuh selanjutnya (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Apabila berhasil, maka *backend* akan memberikan *response code* 200 dengan *body* berisi kondisi *board* setelah *move* tersebut. Langkah ini dilakukan terus-menerus hingga waktu bot habis. Jika waktu bot habis, bot secara otomatis akan dikeluarkan dari *board*.

e. Program *frontend* secara periodik juga akan mengirimkan *GET request* terhadap *endpoint* /api/boards/{id} untuk mendapatkan kondisi *board* terbaru, sehingga tampilan *board* pada *frontend* akan selalu ter-*update*.

2. Implementasi Algoritma Greedy ke Dalam Bot

Untuk menerapkan algoritma yang telah dikembangkan ke dalam bot, langkah pertama adalah membuat folder baru di dalam direktori */game/logic/*, misalnya dengan nama *MyBot.py*. Setelah itu, buatlah sebuah kelas baru yang merupakan turunan dari *BaseLogic*. Dalam kelas ini, perlu diimplementasikan konstruktor (__init__) dan juga metode *next_move*. Berikut ini adalah contoh struktur kelas tersebut :

Fungsi *next_move* dalam contoh sebelumnya harus mengembalikan pasangan nilai *delta_x* dan *delta_y*, dengan pilihan nilai yang valid terbatas pada (1, 0), (0, 1), (-1, 0), atau (0, -1). Jika nilai yang dikembalikan berada di luar daftar tersebut atau bergerak ke posisi di luar *board*, maka pergerakan akan diabaikan oleh sistem dan akan muncul pesan kesalahan berupa *Invalid Move*.

Tahap berikutnya adalah mengimpor kelas yang telah dibuat ke dalam file *main.py*, kemudian menambahkannya ke dalam dictionary *CONTROLLERS*. Berikut ini adalah contoh cara melakukan proses impor tersebut :

```
From game.logic.mybot import Mybot

init()
BASE_URL = "http://localhost:3000/api"
DEFAULT_BOARD_ID = 1
CONTROLLERS = { "Random": Randomlogic, "MyBot": MyBot}
```

Bot yang telah dibuat dapat dijalankan secara individu maupun bersamaan dengan bot lainnya menggunakan *script* *.bat* (untuk Windows) atau *.sh* (untuk Linux/macOS). Cara menjalankan program ini akan dibahas lebih detail pada bagian 3.

3. Menjalankan Program Bot

Program bot yang telah dikembangkan dapat dijalankan melalui aplikasi Command Prompt (CMD) atau terminal. Untuk mengeksekusi satu bot yang menggunakan logika dari file *game/logic/random.py*, digunakan perintah khusus seperti yang ditunjukkan di bawah ini. Perlu diperhatikan bahwa argumen *logic* pada perintah atau *script* tersebut harus disesuaikan dengan nama bot yang telah terdaftar di dalam dictionary *CONTROLLERS*.

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

Untuk menjalankan beberapa bot secara bersamaan, misalnya empat bot yang menggunakan logika yang sama dari *game/logic/random.py*, perlu file terlebih dahulu perlu dibuat sebuah file bernama *run-bots.bat* untuk sistem operasi Windows, atau *run-bots.sh* untuk Linux dan macOS. File tersebut akan berisi sejumlah perintah yang digunakan untuk mengeksekusi keempat bot tersebut secara paralel.

```
@echo off
start cmd /c "python main.py --logic Random
--email=your_email@example.com
--name=your_name --password=your_password --team etimo"
start cmd /c "python main.py --logic Random
--email=your_email@example.com
--name=your_name --password=your_password --team etimo"
start cmd /c "python main.py --logic Random
--email=your_email@example.com
--name=your_name --password=your_password --team etimo"
start cmd /c "python main.py --logic Random
--email=your_email@example.com
--name=your_name --password=your_password --team etimo"
```

Script yang ada pada *run-bots.bat* atau *run-bots.sh* dapat disesuaikan sesuai dengan *logic* file yang digunakan, email, nama, ataupun password.

BAB III
APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Berdasarkan studi pustaka yang telah dilakukan pada landasan teori, yang menyatakan *algoritma greedy* melibatkan pencarian sebuah himpunan bagian (**S)** dari himpunan kandidat (**C**) yang dalam konteks ini, himpunan S harus memenuhi sejumlah kriteria, yakni bahwa S merupakan solusi yang sah dan dioptimalkan berdasarkan suatu fungsi objektif. Oleh karena itu, untuk menerapkan *algoritma greedy* pada permasalahan game *Diamonds*, langkah awal yang perlu dilakukan adalah mengidentifikasi elemen-elemen dalam permainan tersebut dan memetakannya ke dalam komponen-komponen utama dari *algoritma greedy*. Berikut pemetaan yang dilakukan dalam pengembangan *DuhBot* :

1. Himpunan kandidat (C) : Seluruh objek pada *board* seperti *diamond* merah dan biru, *base*, musuh, *base* musuh, teleporter dan *red button*. Setiap objek dinilai berdasarkan posisi dan keuntungannya.
2. Himpunan solusi (S) : Posisi yang memberikan keuntungan lokal maksimal, seperti posisi *diamond* terdekat dengan nilai tinggi yang dapat dijangkau dengan resiko minimal.
3. Fungsi solusi : Mengevaluasi apakah waktu yang tersisa cukup untuk menuju *base* sebelum waktu habis, khususnya jika membawa *diamond*.
4. Fungsi seleksi : Menentukan objek dengan skor tertinggi, dihitung sebagai rasio antara nilai poin *diamond* terhadap jarak dan menghindari zona berbahaya.
5. Fungsi kelayakan : Mengecek apakah *inventory* cukup untuk mengambil *diamond* (misal, tidak mengambil *diamond* merah saat hanya tersisa satu slot.
6. Fungsi objektif : Memaksimalkan skor akhir dengan memilih langkah yang memberikan hasil lokal terbaik.

3.2 Eksplorasi Alternatif Solusi Greedy

Dalam tahap eksplorasi, Kami telah dilakukan pertimbangan terhadap berbagai pendekatan *greedy* :

1. Greedy Direct Diamond
    - Langsung memilih diamond dengan skor tertinggi (nilai/jarak)
    - Sangat sederhana dan cepat
2. Greedy Tackle
    - Menyerang bot musuh jika mereka membawa *diamond* dan jaraknya <= 2
    - Hanya dilakukan jika DuhBot tidak sedang membawa diamond
3. Greedy Avoid Enemy
    - Menghindar dari musuh jika sedang membawa *diamond*
    - Menentukan arah gerak menjauh dari musuh
4. Greedy Teleporter
    - Menggunakan BFS untuk mencari teleporter dan menghitung apakah teleportasi mempercepat perjalanan
5. Greedy Red Button
    - Menargetkan tombol merah jika jumlah *diamond* sedikit dan tombol berada dalam radius aman.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Berdasarkan algoritma yang telah dirancang, dilakukan serangkaian pengujian untuk mengevaluasi sejauh mana efektivitas dan kinerja algoritma tersebut. Pengujian ini akan dilaksanakan melalui beberapa tahapan atau proses sebagai berikut :

1. Greedy Direct Diamond memiliki performa tertinggi dalam simulasi single-player, dengan rata-rata 10 sampai 30 point per game.
2. Greedy Tackle efektif dalam kondisi 1v1, namun tidak stabil di multiplayer karena tergantung pada keberadaan musuh.
3. Greedy Avoid Enemy terbukti mencegah kehilangan *diamond*, menjaga efisiensi jangka panjang.
4. Teleporter menurunkan waktu tempuh jika digunakan dalam radius optimal.
5. Red Button meningkatkan efisiensi saat *board* dalam keadaan sepi.

3.4 Strategi Greedy yang Dipilih

Setelah melalui pengujian dan evaluasi, strategi utama *DuhBot* terdiri dari kombinasi berikut :

1. Greedy Direct Diamond sebagai strategi utama untuk memilih *diamond* terbaik.
2. Greedy Safe Return : Kembali ke *base* saat *inventory* penuh atau waktu <10s.
3. Greedy Avoid Enemy : Menghindar dari musuh jika membawa *diamond*.
4. Greedy Tackle : Menyerang jika musuh membawa *diamond* dan *DuhBot* tidak membawa apa-apa.
5. Greedy Teleporter dan Red Button : Digunakan secara kondisional.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

```
# Aditya (123140093) & Nabila (123140099)
# Kelompok 6 - DuhBot | Anak Pak Imam
# Nama DuhBot dipilih karena menggambarkan reaksi spontan "duh" yang sering
muncul saat menemukan masalah tak terduga dalam algoritma
# Serta "bot" yang mewakili pendekatan berbasis logika dan otomasi.
# Nama ini juga mencerminkan sifat bot yang terkadang membuat keputusan yang
tidak terduga atau lucu atau bahkan melakukan kesalahan.

from game.logic.base import BaseLogic
from game.models import Board, GameObject
from collections import deque
import random

class DuhBot(BaseLogic):
    def __init__(self):
        super().__init__()
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.status = "BERBURU"
        self.last_direction = None
        self.danger_zone = []
        self.data_enemy = {}

    def hitung_jarak(self, pos1, pos2):
        """Menghitung jarak antara dua titik (Manhattan distance)"""
        return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)

    def menentukan_arah(self, pos_sekarang, pos_tujuan):
        """Memilih arah menuju tujuan"""
        selisih_x = pos_tujuan.x - pos_sekarang.x
        selisih_y = pos_tujuan.y - pos_sekarang.y

        if abs(selisih_x) > abs(selisih_y):
            return (1 if selisih_x > 0 else -1, 0)  # memprioritaskan gerak
horizontal
        else:
            return (0, 1 if selisih_y > 0 else -1)  # memprioritaskan gerak
vertikal

    def cari_teleport(self, board, pos_sekarang):
```

```
        """Mencari teleporter terdekat menggunakan BFS"""
        if not hasattr(board, 'teleporters') or len(board.teleporters) < 2:
            return None

        antrian = deque([(pos_sekarang.x, pos_sekarang.y, [])])
        sudah_dilewati = set()
        sudah_dilewati.add((pos_sekarang.x, pos_sekarang.y))

        while antrian:
            x, y, jalur = antrian.popleft()

            for tele in board.teleporters:
                if (x, y) == (tele.position.x, tele.position.y):
                    return tele.position, jalur[0] if jalur else None

            for dx, dy in self.directions:
                x_baru, y_baru = x + dx, y + dy
                if (0 <= x_baru < board.width and
                    0 <= y_baru < board.height and
                    (x_baru, y_baru) not in sudah_dilewati and
                    (x_baru, y_baru) not in self.danger_zone):
                    sudah_dilewati.add((x_baru, y_baru))
                    antrian.append((x_baru, y_baru, jalur + [(dx, dy)]))
        return None

    def harus_menyerang(self, bot_saya, board):
        """Memutuskan apakah akan menyerang bot lawan"""
        if bot_saya.properties.diamonds == 0:  # hanya menyerang jika tidak
membawa berlian
            for enemy in board.bots:
                if (enemy.id != bot_saya.id and
                    enemy.properties.diamonds > 0 and
                    self.hitung_jarak(bot_saya.position, enemy.position) <= 2):
                    return True
        return False

    def next_move(self, bot_saya: GameObject, board: Board):
        base = bot_saya.properties.base
        berlian_dibawa = bot_saya.properties.diamonds
        posisi_sekarang = bot_saya.position
        waktu_tersisa = bot_saya.properties.milliseconds_left / 1000
        self.danger_zone = []

        # 1. mode cepat pulang jika waktu tersisa kurang dari 10 detik
        if waktu_tersisa < 10 and berlian_dibawa > 0:
            waktu_ke_base = self.hitung_jarak(posisi_sekarang, base) * 0.3
            if waktu_tersisa < waktu_ke_base + 1:
```

```
                self.status = "CEPAT_PULANG"
                return self.menentukan_arah(posisi_sekarang, base)

        # 2. mode menyerang lawan
        if self.harus_menyerang(bot_saya, board):
            for enemy in board.bots:
                if (enemy.id != bot_saya.id and
                    enemy.properties.diamonds > 0 and
                    self.hitung_jarak(posisi_sekarang, enemy.position) <= 2):
                    self.status = "SERANG"
                    return self.menentukan_arah(posisi_sekarang,
enemy.position)

        # 3. kembali ke base jika membawa berlian
        if (berlian_dibawa >= 5 or
            (berlian_dibawa == 4 and any(d.properties.points == 2 for d in
board.diamonds)) or
            (berlian_dibawa >= 3 and waktu_tersisa < 20)):
            self.status = "PULANG"
            return self.menentukan_arah(posisi_sekarang, base)

        # 4. menghindari bot lawan
        if berlian_dibawa > 0:
            for enemy in board.bots:
                if (enemy.id != bot_saya.id and
                    self.hitung_jarak(posisi_sekarang, enemy.position) <= 2):
                    self.status = "MENGHINDAR"
                    self.danger_zone.append((enemy.position.x,
enemy.position.y))
                    selisih_x = posisi_sekarang.x - enemy.position.x
                    selisih_y = posisi_sekarang.y - enemy.position.y
                    arah = (1 if selisih_x < 0 else -1, 0) if abs(selisih_x) >
abs(selisih_y) else (0, 1 if selisih_y < 0 else -1)
                    return arah

        # 5. memanfaatkan teleportasi
        info_teleport = self.cari_teleport(board, posisi_sekarang)
        if info_teleport and self.status != "MENGHINDAR":
            posisi_tele, jalur = info_teleport
            if jalur and self.hitung_jarak(posisi_sekarang, posisi_tele) < 4:
                return jalur

        # 6. memilih berlian terbaik
        nilai_tertinggi = -float('inf')
        berlian_terbaik = None

        for diamond in board.diamonds:
```

```
            jarak = self.hitung_jarak(posisi_sekarang, diamond.position)
            nilai_berlian = diamond.properties.points

            if berlian_dibawa + nilai_berlian > 5:
                continue

            skor = nilai_berlian / (jarak + 1)

            if nilai_berlian == 2:
                skor *= 1.5

            if (diamond.position.x, diamond.position.y) in self.danger_zone:
                skor *= 0.3

            if skor > nilai_tertinggi:
                nilai_tertinggi = skor
                berlian_terbaik = diamond

        if berlian_terbaik:
            self.status = "MENGUMPULKAN"
            return self.menentukan_arah(posisi_sekarang,
berlian_terbaik.position)

        # 7. mencari tombol untuk teleportasi
        if hasattr(board, 'buttons'):
            for button in board.buttons:
                if (self.hitung_jarak(posisi_sekarang, button.position) < 3 and
                    (button.position.x, button.position.y) not in
self.danger_zone):
                    return self.menentukan_arah(posisi_sekarang,
button.position)

        # 8. kembali ke base jika membawa berlian
        if berlian_dibawa > 0:
            self.status = "KEMBALI"
            return self.menentukan_arah(posisi_sekarang, base)

        # 9. gerak acak jika tidak ada pilihan lain
        arah_mungkin = [
            (dx, dy) for dx, dy in self.directions
            if (0 <= posisi_sekarang.x + dx < board.width and
                0 <= posisi_sekarang.y + dy < board.height and
                (dx, dy) != (0, 0))
        ]
        if arah_mungkin:
            return random.choice(arah_mungkin)
        return (0, 0)
```

2. Struktur Data yang Digunakan

Program ini menggunakan beberapa struktur data dari library game dan juga beberapa variabel yang untuk mendukung logika pengambilan keputusan bot. Berikut rincian struktur data dan penggunaannya :

1. Position

```
class Position :
    X : int
    Y : int
```

Digunakan untuk menyimpan informasi posisi dalam game berupa koordinat x dan y. Penggunaan dalam kode yaitu :

- posisi_sekarang = bot_saya.position
- self.hitung_jarak(pos1, pos2) menerima parameter bertipe position

2. GameObject dan Properties

```
class GameObject :
    id : int
    Position : Position
    type :str
    Properties : Properties
```

Kelas GameObject mewakili objek apapun dalam permainan : bot, diamond, base, teleporter dan red button. Masing-masing GameObject memiliki :

Digunakan untuk menyimpan informasi posisi dalam game berupa koordinat x dan y. Penggunaan dalam kode yaitu :

- position : posisi objek (tipe Position)
- type : jenis objek dalam bentuk string
- properties : properti tambahan (tipe Properties) tergantung pada jenis objek

Contoh penggunaannya, yaitu :

- bot_saya.properties.diamonds
- enemy.position.x
- diamond.properties.points

```
class Properties :
    points : int
    Pair_id : str
    diamonds : int
    score : int
```

```
name : str
iventory_size : int
can_tackle : bool
milliseconds_left : int
time_joined : str
base : position
```

3. Variabel yang Digunakan dalam Bot

```
a. danger_zone : list
b. self.directions : list of tuple
c. self.status : str
d. self.last_direction : tuple
e. self.data_enemy : dict
```

Penjelasan yang dalam tabel :

a. List posisi dianggap berbahaya oleh bot, biasanya berisi posisi bot musuh jika sedang membawa berlian.
b. Menyimpan arah gerak yang mungkin dalam bentuk (dx, dy).
c. Menunjukkan status/strategi bot saat ini, misalnya “BERBURU”, “CEPAT_PULANG”
d. Menyimpan arah terakhir yang diambil oleh bot. Dalam kode ini belum banyak dimanfaatkan.
e. Tempat menyimpan informasi tambahan tentang musuh. Namun belum digunakan dalam kode ini tapi disiapkan untuk ekspansi strategi.

4.3 Pengujian Program

1. Skenario Pengujian

Pengujian dilakukan dalam dua skenario utama, yaitu :

a. Skenario 1 : Mode Single Player yang bertujuan untuk mengukur efektivitas *Greedy Direct Diamond* dan *Safe Return* dalam mengumpulkan poin tanpa gangguan musuh. Yang dimana *DuhBot* bermain sendiri pada papan dengan distribusi *diamond* acak.
b. Skenario 2 : Mode Multiplayer (1 vs 1 vs banyak) yang bertujuan mengukur ketahanan dan kecerdasan taktik *Greedy Tackle* dan *Greedy Avoid Enemy* saat berinteraksi dengan bot lawan. Yang dimana kondisi *DuhBot* dipasangkan dengan bot lain pada yang sama.

2. Hasil Pengujian dan Analisis

a. Skenario 1 : *DuhBot* bergerak sendiri dengan rata-rata poin yang dihasilkan 20 poin. Strategi Direct Diamond berjalan optimal. *DuhBot* mampu mengumpulkan *diamond* secara efisien dan pulang tepat waktu.

b. Skenario 2 : *DuhBot* melawan 1 bot dengan rata-rata poin yang dihasilkan 16 poin. *DuhBot* efektif menyerang jika lawan membawa *diamond* dan *DuhBot* tidak membawa apa-apa. Strategi *Greedy Tackle* menambah poin sekitar 1-2 dari hasil tackle.

c. Skenario 2B : *DuhBot* melawan 3 bot dengan rata-rata poin yang dihasilkan 5 poin. Strategi *Avoid Enemy* bekerja dengan baik tetapi banyak *diamond* direbut lawan sementara *DuhBot* lebih defensif. *Red Button* dan Teleporter dimanfaatkan saat situasi sepi.

Analisis :

a. Strategi utama *Greedy Direct Diamond* menunjukkan hasil konsisten dalam kondisi stabil

b. Kombinasi *Avoid Enemy* dan *Safe Return* mampu meminimalkan kehilangan, tetapi menurunkan agresivitas.

c. *Greedy Tackle* hanya efektif saat lawan membawa *diamond*, posisi dekat dan *DuhBot* tidak membawa apa-apa, sehingga penggunaannya bersifat situasional.

d. *Greedy Teleporter* dan *Red Button* lebih berguna dalam kondisi peta sepi atau posisi tidak menguntungkan.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Pengembangan *DuhBot* berhasil membuktikan bahwa *algoritma greedy* dapat diterapkan secara efektif dalam permainan berbasis strategi seperti *Diamondsi*. Berbagai pendekatan *greedy* yang digunakan seperti *Direct Diamond, Avoid Enemy* dan *Safe Return* mampu berfungsi sesuai dengan harapan dalam sebagian besar skenario pengujian. Hasil yang diperoleh menunjukkan bahwa *DuhBot* cukup adaptif dalam mengejar skor maksimum serta menghindari resiko kehilangan *diamond*.

Namun, performa bot masih dapat ditingkatkan, terutama dalam situasi multiplayer kompleks serta pengembangan bot ini masih belum maksimal. Awalnya kami sudah membuat 2 bot yang berbeda dengan penerapan strategi *greedy* yang berbeda, namun bot belum cukup efektif dan perlu perbaikan lagi sehingga tidak jadi menggunakan dan mencoba bot tersebut. Hal ini disebabkan oleh waktu eksplorasi dan pengujian yang belum cukup optimal, bukan karena kurangnya perhatian terhadap pengerjaan tugas ini, melainkan karena fokus dan waktu tim terbagi untuk menyelesaikan beberapa tugas lain dari mata kuliah yang berbeda yang memiliki tenggat waktu hampir bersamaan. Meski dalam kondisi tersebut, tugas ini tetap dikerjakan dengan sepenuh hati, melalui proses pemahaman ulang materi strategi algoritma, eksplorasi pendekatan yang relevan dan usaha maksimal dalam setiap perancangan serta implementasi.

5.2 Saran

Disarankan untuk merancang berbagai skenario permainan secara sistematis guna mengidentifikasi potensi kekurangan, kelemahan strategi maupun error teknis yang mungkin muncul. Dengan pengujian yang lebih luas dan terstruktur, solusi-solusi alternatif yang lebih optimal dan efektif dapat ditemukan. Penerapan langkah-langkah tersebut diharapkan mampu meningkatkan performa *algoritma greedy* dalam menyelesaikan permainan *Diamonds* secara lebih kompetitif dan adaptif.

LAMPIRAN

A. Repository Github (*GitHub DuhBot*)

DAFTAR PUSTAKA

[1] R. Munir, Rinaldi. 2025. Algoritma Greedy (Bagian 1), Sekolah Teknik Elektro dan Informatika,https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-Greedy-(2025)-Bag1.pdf