

LAPORAN TUGAS BESAR

IF2211 Strategi Algoritma



Pemanfaatan Algoritma *Greedy* dalam Pembuatan *Bot* Permainan Diamonds

Dipersiapkan oleh:

Kelompok “Robotnik” (K02)

1. Rici Trisna Putra (13522026)
2. Imam Hanif Mulyarahman (13522030)
3. Diero Arga Purnama (13522056)

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung,
Jl. Ganesha 10, Bandung 40132

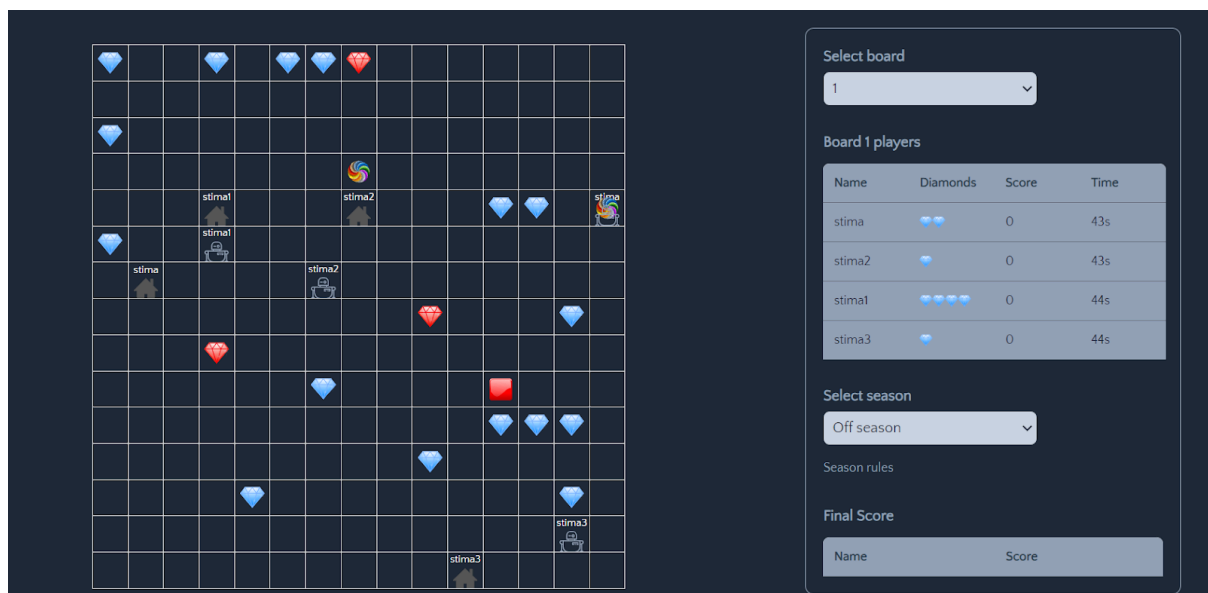
DAFTAR ISI

DAFTAR ISI	1
BAB I	2
DESKRIPSI TUGAS	2
BAB II	6
LANDASAN TEORI	6
2.1 Algoritma Greedy.....	6
2.2 Cara Kerja Program.....	6
BAB 3	8
APLIKASI STRATEGI GREEDY	8
3.1 Mapping persoalan Diamonds.....	8
3.2 Eksplorasi Alternatif Solusi.....	9
3.3 Analisis Efisiensi dan Efektivitas.....	9
3.4 Strategi Greedy yang Dipilih.....	9
BAB 4	10
IMPLEMENTASI DAN PENGUJIAN	10
4.1 Implementasi Algoritma Greedy.....	10
4.2 Penjelasan Struktur Data.....	17
4.3 Analisis Desain Solusi.....	17
BAB 5	20
KESIMPULAN DAN SARAN	20
5.1 Kesimpulan.....	20
5.2 Saran.....	20
LAMPIRAN	21
DAFTAR PUSTAKA	22

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. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.



Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya mahasiswa harus menggunakan **strategi greedy** dalam membuat bot ini.

Program permainan Diamonds terdiri atas:

1. *Game engine*, yang secara umum berisi:
 - a. Kode *backend* permainan, yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *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 kalian implementasikan dengan algoritma *greedy* untuk bot kelompok kalian)
 - c. Program utama (*main*) dan utilitas lainnya

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan *game engine* dan membuat bot dari *bot starter pack* yang telah tersedia pada pranala berikut.

- **Game engine :**
<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>
- **Bot starter pack :**
<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>

Komponen-komponen dari permainan Diamonds antara lain:

1. **Diamonds**



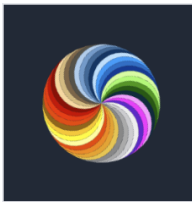
Untuk memenangkan pertandingan, kita harus mengumpulkan *diamond* ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis *diamond* yaitu *diamond* biru dan *diamond* merah. *Diamond* merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. *Diamond* akan di-*regenerate* secara berkala dan rasio antara *diamond* merah dan biru ini akan berubah setiap *regeneration*.

2. **Red Button/Diamond Button**



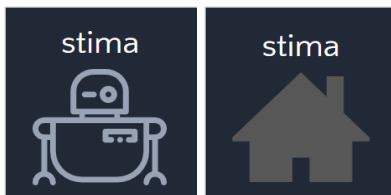
Ketika *red button* ini dilewati/dilangkahi, semua *diamond* (termasuk *red diamond*) akan di-*generate* kembali pada *board* dengan posisi acak. Posisi *red button* ini juga akan berubah secara acak jika *red button* ini dilangkahi.

3. Teleporters



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



Pada game ini 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* (akan dijelaskan di bawah) bot menjadi kosong.

5. Inventory

The screenshot shows a game interface with a 10x10 grid. Various icons are placed on the grid, including blue diamonds, red diamonds, a colorful spiral (teleporter), and robot icons labeled 'stima1', 'stima2', and 'stima3'. Some robot icons are accompanied by house icons (bases). The sidebar on the right contains the following sections:

- Select board:** A dropdown menu showing '1'.
- Board 1 players:** A table with 4 columns: Name, Diamonds, Score, and Time.
- Select season:** A dropdown menu showing 'Off season'.
- Season rules:** A section for rules.
- Final Score:** A table with 2 columns: Name and Score.

Name	Diamonds	Score	Time
stima	2	0	43s
stima2	1	0	43s
stima1	4	0	44s
stima3	1	0	44s

Name	Score
stima	0
stima2	0
stima1	0
stima3	0

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.

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.

BAB II

LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma *greedy* adalah algoritma yang digunakan untuk mencari solusi dari sebuah masalah secara langkah per langkah. Pada setiap langkahnya, algoritma *greedy* akan mengambil pilihan terbaik yang dapat diambilnya pada saat itu tanpa memperhatikan bagaimana langkah tersebut akan mempengaruhi langkah berikutnya. Solusi yang didapatkan dari algoritma *greedy* tidak selalu merupakan solusi yang optimal.

2.2 Cara Kerja Program

Cara bot kelompok kami melakukan aksinya mengikuti template yang terdapat dalam `base.py`, dimana bot akan mengembalikan arah gerakannya melalui `method next_move`. `Method next_move` akan menentukan arah gerakan bot melalui beberapa langkah, yaitu:

1. Mengecek apakah bot sudah memiliki *goal*.
2. Jika bot belum memiliki *goal* maka akan digunakan sebuah algoritma untuk menentukan *goal* bot.
3. Jika bot sudah memiliki *goal* maka fungsi akan langsung mengembalikan arah dari *goal* tersebut.

Algoritma yang akan digunakan untuk menentukan *goal* / tujuan dari bot adalah algoritma *greedy*. Versi dari algoritma *greedy* yang kami gunakan adalah *Greedy by density* yaitu pemilihan objek yang memiliki nilai poin per jarak terbesar.

Implementasi algoritma *greedy* ke dalam bot dilakukan melalui fungsi seleksi yang terdapat dalam `method next_move`. Di dalam fungsi tersebut, himpunan kandidat (list yang memuat seluruh *diamond* yang terdapat dalam *board*) akan diproses sampai dihasilkan list yang berisi nilai poin per jarak dari tiap-tiap *diamond* yang ada. Setelah didapatkan list tersebut, akan diambil *diamond* yang memiliki nilai terbesar yang kemudian akan dicek kelayakannya oleh fungsi layak, jika *diamond* yang didapatkan adalah layak, maka *diamond* tersebut akan dikembalikan oleh fungsi seleksi.

Game engine yang digunakan adalah game engine yang sudah disediakan dalam folder game engine. Adapun program untuk menjalankan bot juga sudah disediakan dalam bot starter pack. Langkah-langkah untuk menjalankan game engine adalah sebagai berikut:

1. Install semua requirement yang diperlukan.
2. Instalasi dan konfigurasi awal.
3. Build melalui command **npm run build**.
4. Jalankan dengan command **npm run start**.

Adapun cara untuk menjalankan bot adalah sebagai berikut:

1. Install semua requirement yang diperlukan.
2. Instalasi dan konfigurasi awal.
3. Jalankan dengan mengetikkan command.

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

Bagian logic, email, nama, dan password (bagian yang berbentuk miring) dapat diganti sesuai kebutuhan.

BAB 3

APLIKASI STRATEGI *GREEDY*

3.1 Mapping persoalan Diamonds

Algoritma *greedy* memiliki 6 elemen, yaitu:

1. Himpunan kandidat, C: Berisi kandidat yang dapat dipilih tiap langkah.
2. Himpunan solusi, S: Berisi kandidat yang sudah dipilih
3. Fungsi solusi: Menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi seleksi: Memilih kandidat berdasarkan strategi *greedy* tertentu. Strategi ini bersifat heuristik.
5. Fungsi kelayakan: Memeriksa apakah kandidat yang dipilih layak dimasukkan ke dalam himpunan solusi.
6. Fungsi obyektif: Memaksimumkan atau meminimumkan.

No	Objek	Kegunaan	Fungsi Proses
1	Diamond	<ul style="list-style-type: none"> Menambah poin pada inventory sesuai dengan bobot tiap diamond. Di handle dengan <i>greedy by density</i> yakni <i>greedy by distance</i> yang mempertimbangkan bobot diamond. 	seleksi
2	Reset Button	<ul style="list-style-type: none"> Mengacak semua diamond dan mengenerasi diamond baru pada board. Tidak di handle karena dirasa pemborosan langkah. 	-
3	Teleporter	<ul style="list-style-type: none"> Membuat bot yang melewati teleporter untuk 	seleksi, JarakTeleporter
4	Bot dan Base	<ul style="list-style-type: none"> Bot akan mengumpulkan diamond sebanyak-banyaknya kemudian kembali ke base. 	next_move
5	Inventory	<ul style="list-style-type: none"> Tempat penyimpanan sementara diamond yang diambil oleh bot 	fungsiSolusi

3.2 Eksplorasi Alternatif Solusi

Menurut kelompok kami, terdapat beberapa alternatif solusi greedy yang dapat digunakan dalam menyelesaikan persoalan optimasi jalur bot. Alternatif pertama yang kami usulkan adalah greedy by distance from bot. Algoritma greedy by distance from bot ini akan selalu memprioritaskan diamond terdekat dari posisi bot tanpa mempertimbangkan hal lain. Alternatif kedua adalah greedy by distance from base. Alternatif ini akan selalu memprioritaskan diamond dengan jarak yang paling dekat dengan base. Alternatif ketiga adalah algoritma greedy by point diamond. Algoritma greedy ini mencari diamond dengan poin tertinggi sebagai prioritas utama. Alternatif keempat adalah algoritma greedy by point yang menganggap reset button sebagai diamond. Algoritma ini akan mengambil diamond dengan poin tertinggi sekaligus mempertimbangkan untuk mereset board apabila reset button merupakan objek terdekat dari bot tersebut.

3.3 Analisis Efisiensi dan Efektivitas

Pada Alternatif pertama, yaitu greedy by distance from bot yang tidak mempertimbangkan hal lain, memiliki keuntungan apabila diamond berdekatan dengan posisi bot. Namun, alternatif ini kurang didukung disebabkan terlalu simpel karena tidak memperhitungkan faktor-faktor lain yang terdapat dalam permainan. Alternatif selanjutnya, yaitu greedy by distance from base, memiliki keuntungan apabila banyak diamond yang berdekatan dengan base, tetapi memiliki kelemahan bot akan sering berputar di sekitar base apabila ada diamond terdekat berada di wilayah yang jauh dengan bot tetapi dekat dari base. Alternatif berikutnya, yaitu algoritma greedy by point diamond, memiliki keuntungan apabila terdapat banyak diamond merah berdekatan, tetapi sangat tidak efisien karena tidak banyak diamond merah yang muncul di map dan diamond merah yang juga mengambil porsi inventory yang besar. Alternatif keempat yaitu greedy by point dengan mempertimbangkan keberadaan red button. Alternatif ini tidak dipilih karena dirasa hanya akan membuang move apabila mengejar red button. Selain itu, menekan red button juga malah dapat menguntungkan bot milik lawan.

3.4 Strategi Greedy yang Dipilih

Strategi *Greedy* yang kami pilih adalah *greedy by density* dengan hanya mempertimbangkan teleporter. Algoritma greedy ini dipilih karena merupakan algoritma yang paling fleksibel dalam mempertimbangkan berbagai situasi. Algoritma ini merupakan pengembangan dari algoritma greedy by distant dan greedy by point dengan mempertimbangkan keberadaan teleporter pada permainan. Pada algoritma ini akan diperiksa jarak terhadap bot dan besar point dari suatu diamond dan membandingkannya dengan diamond lainnya. Keberadaan teleporter juga dapat mempengaruhi besar jarak yang akan dihasilkan. Dengan kata lain algoritma greedy by density akan mempertimbangkan jarak dan poin untuk mendapatkan rute terbaik untuk mendapat poin sebanyak-banyaknya.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

Implementasi algoritma dalam bentuk pseudocode

```
class PenyuTELEPORTER(BaseLogic):

    procedure __init__(self)

        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]

        self.goal_position: Optional[Position] = None

        self.poin_lama: int = 0

    function next_move(self, board_bot: GameObject, board: Board) -> (int,int)

        # FUNGSI PEMBANTU

        function isEqualPosition(Pos1, Pos2) -> boolean

            # Mengembalikan true jika Pos1 sama dengan Pos2

            if not(Pos1 and Pos2) then

                return False

            return (Pos1.x == Pos2.x and Pos1.y == Pos2.y)

        function isTargetValid(diamonds, teleporters) -> boolean

            # Mengembalikan true jika terdapat diamond pada posisi self.goal_position

            return (any(isEqualPosition(diamond.position, self.goal_position) for
            diamond in diamonds) or any(isEqualPosition(teleporter.position,
            self.goal_position) for teleporter in teleporters))

        function isTeleporter(board_bot: GameObject, teleporters: List[GameObject])
        -> boolean:

            # Mengembalikan true jika terdapat teleporter pada posisi bot

            return (any(isEqualPosition(teleporter.position, board_bot.position) for
```

```
teleporter in teleporters))
```

```
function Jarak(Pos1: Position, Pos2: Position) -> integer
```

```
    # Mengembalikan jumlah gerakan yang diperlukan untuk bergerak dari  
    Pos1 ke Pos2
```

```
    return (abs(Pos1.y - Pos2.y) + abs(Pos1.x - Pos2.x))
```

```
function JarakTeleporter(board_bot: GameObject, teleporters:  
List[GameObject], diamond: Optional[GameObject], base: Optional[Position]) ->  
(GameObject, integer)
```

```
    nearest_teleporter = teleporters[0]
```

```
    farthest_teleporter = teleporters[1]
```

```
    current_position = board_bot.position
```

```
    if Jarak(current_position, teleporters[1].position) < Jarak(current_position,  
teleporters[0].position) then
```

```
        nearest_teleporter = teleporters[1]
```

```
        farthest_teleporter = teleporters[0]
```

```
    if (base) then
```

```
        return nearest_teleporter, Jarak(current_position,  
nearest_teleporter.position) + Jarak(base, farthest_teleporter.position)
```

```
        return nearest_teleporter, Jarak(current_position,  
nearest_teleporter.position) + Jarak(diamond.position, farthest_teleporter.position)
```

```
# FUNGSI SELEKSI
```

```
function seleksi(diamonds: List[GameObject], board_bot: GameObject, board:  
Board) -> (Optional[GameObject], Optional[GameObject]):
```

```
    # Mengembalikan diamond atau none jika tidak terdapat diamond yang  
    memenuhi fungsi kelayakan
```

```
# FUNGSI KELAYAKAN
```

```
function layak(kandidatSolusi: (Optional[GameObject],  
Optional[GameObject]), board_bot: GameObject) -> boolean:
```

```

# Mengembalikan true jika diamond yang dipilih layak

if kandidatSolusi[0] then
    if isEqualPosition(board_bot.position, kandidatSolusi[0].position) then
        return False

    if kandidatSolusi[1] then
        if isEqualPosition(board_bot.position, kandidatSolusi[1].position) then
            return False

        if (kandidatSolusi[1].properties.points == 2 and
board_bot.properties.diamonds >= 4) then
            return False

        # return (board_bot.properties.diamonds +
kandidatSolusi.properties.points <= 5)

        return True

function rumus(n: GameObject, isTeleporter: bool, teleporters) -> float
    if (isTeleporter) then
        nearest_teleporter, dist = JarakTeleporter(board_bot, teleporters, n,
None)

        return (nearest_teleporter, (dist / n.properties.points))

    return ((Jarak(current_position, n.position) / n.properties.points))

teleporters = [t for t in board.game_objects if t.type ==
"TeleportGameObject"]

# list poin diamond dengan jarak tanpa melalui teleporter
poin = [rumus(n, False, teleporters) for n in diamonds]

# list teleporter terdekat dan poin diamond dengan jarak yang dihitung
melalui teleporter
poin_teleporter = [rumus(n, True, teleporters) for n in diamonds]

```

```

# jarak minimum diamond tanpa melalui teleporter
min_nt = min(poin)
index_min = poin.index(min_nt)

# cari min + index min dari poin_teleporter
index_min_tel, index = 0, 0
min_teleporter = poin_teleporter[0][1]
for i in poin_teleporter:
    if i[1] < min_teleporter then
        min_teleporter = i[1]
        index_min_tel = index
    index += 1

# membandingkan jarak terdekat melalui teleporter dengan jarak terdekat
tanpa melalui teleporter
isTeleporter = False
if (min_teleporter < min_nt) then
    index_min = index_min_tel
    isTeleporter = True # apabila jarak terdekat melalui teleporter maka
target teleporter terdekat

kandidatSolusi = poin_teleporter[index_min][0] if isTeleporter else None,
diamonds[index_min]

# mengecek kelayakan kandidat solusi
while not layak(kandidatSolusi, board_bot) do
    if len(diamonds) - 1 == 0 then
        # Tidak ada diamond yang layak, balik ke base

```

```

        kandidatSolusi = None, None

        break

    del diamonds[index_min]

    kandidatSolusi = seleksi(diamonds, board_bot, board)

return (kandidatSolusi)

# FUNGSI OBYEKTIF

function resetTarget(current_position: Position, stats: Properties,
teleporters: List[GameObject]) -> boolean:

    # Mengembalikan true jika:

    if isEqualPosition(current_position, stats.base) then

        # Ada di base

        return True

    if isTeleporter(board_bot, teleporters) then

        # Masuk teleporter

        return True

    if self.goal_position then

        if isEqualPosition(self.goal_position, current_position) then

            # Sudah sampai tujuan

            return True

        if not isEqualPosition(self.goal_position, stats.base) and not
isTargetValid(diamonds, teleporters) then

            # Target yang dituju sudah tidak ada di board

            return True

    if self.poin_lama != stats.diamonds then

        # Poin berubah

        return True

```

```

        return False

# FUNGSI SOLUSI

function fungsiSolusi(stats: Properties) -> boolean:
    return (stats.diamonds == 5)

#-----#

    stats = board_bot.properties
    current_position = board_bot.position
    diamonds = board.diamonds
    teleporters = [t for t in board.game_objects if t.type == "TeleportGameObject"]

    if resetTarget(current_position, stats, teleporters) then
        self.goal_position = None

    if fungsiSolusi(stats) then
        # apabila inventory penuh kembali ke base
        self.goal_position = stats.base
    else if stats.diamonds != 5 and self.goal_position == None then
        solusi = seleksi(diamonds, board_bot, board)
        self.poin_lama = board_bot.properties.diamonds

        if not solusi[1] then
            self.goal_position = stats.base
        else
            self.goal_position = solusi[1].position if not solusi[0] else
solusi[0].position

    if(self.goal_position == stats.base) then

```



```

        nearest_teleporter, distTeleporter = JarakTeleporter(board_bot, teleporters,
None, stats.base)

        distNormal = Jarak(current_position, stats.base)

        if( distTeleporter < distNormal) then
            self.goal_position = nearest_teleporter.position
        else
            self.goal_position = stats.base

    delta_x, delta_y = get_direction(
        current_position.x,
        current_position.y,
        self.goal_position.x,
        self.goal_position.y,
    )

    if (delta_x == delta_y) then
        for i in self.directions:
            if 0 <= current_position.x + i[0] <= 14 and 0 <= current_position.y + i[1]
<= 14:
                delta_x = i[0]
                delta_y = i[1]
                break

            # delta_x, delta_y = self.next_move(board_bot, board)

    return delta_x, delta_y

# Himpunan Kandidat : diamonds

```

```
# Himpunan Solusi : self.goal_position ()  
# Fungsi solusi : solusi()  
# Fungsi seleksi : seleksi()  
# Fungsi kelayakan : layak()  
# Fungsi objektif : resetTarget()
```

4.2 Penjelasan Struktur Data

Bot kami memiliki dua method, yaitu `__init__` dan `next_move`, di dalam method `next_move()`, kami menggunakan fungsi-fungsi berikut:

- `isEqualPosition`: Menentukan apakah dua posisi merupakan posisi yang sama atau tidak.
- `isTargetValid`: Menentukan apakah target bot valid atau tidak.
- `isTeleporter`: Menentukan apakah robot berada dalam teleporter atau tidak.
- `Jarak`: Menghasilkan jarak antara dua posisi.
- `JarakTeleporter`: Menghasilkan teleporter terdekat dan jarak bot dengan suatu objek melalui teleporter.
- `seleksi`: Mengembalikan diamond yang dipilih melalui algoritma *greedy*.
- `layak`: Mengembalikan true jika diamond yang dipilih layak.
- `rumus`: Mengembalikan poin diamond berdasarkan rumus jarak per poin.

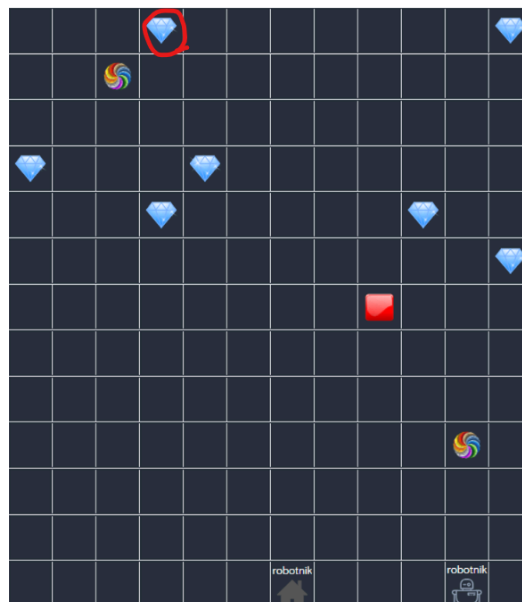
4.3 Analisis Desain Solusi



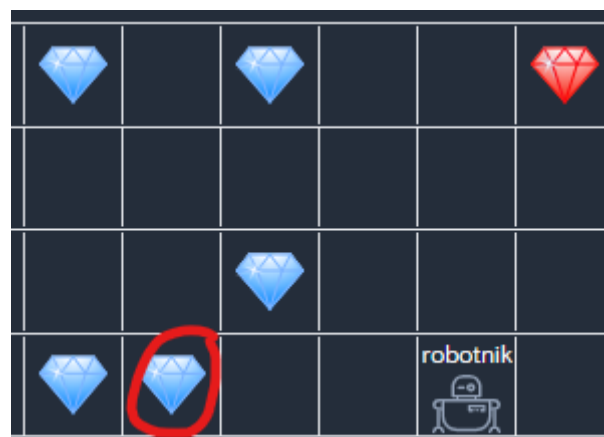
Pada kondisi ini, bot akan memilih diamond merah yang dilingkari merah. Solusi ini optimal, karena algoritma *greedy* yang kita gunakan mempertimbangkan perbandingan antara jarak dengan poin diamond.



Pada kondisi ini, bot akan memilih diamond biru yang dilingkari merah. Solusi ini optimal, karena jika semua diamond memiliki jumlah poin yang sama, diamond yang dipilih adalah diamond yang terdekat.

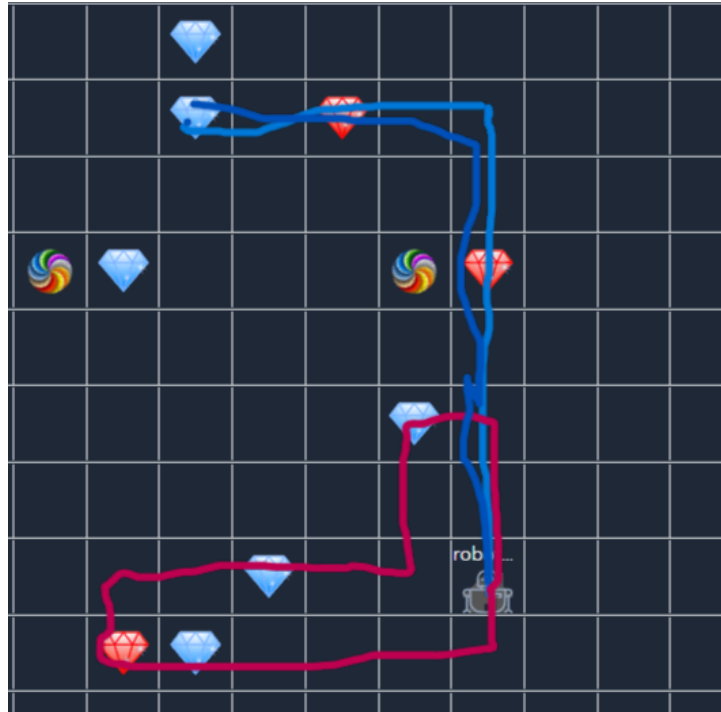


Pada kondisi ini, bot akan memilih diamond biru yang dilingkari merah. Solusi ini optimal, karena diamond tersebut paling dekat dengan robot jika menggunakan teleporter.

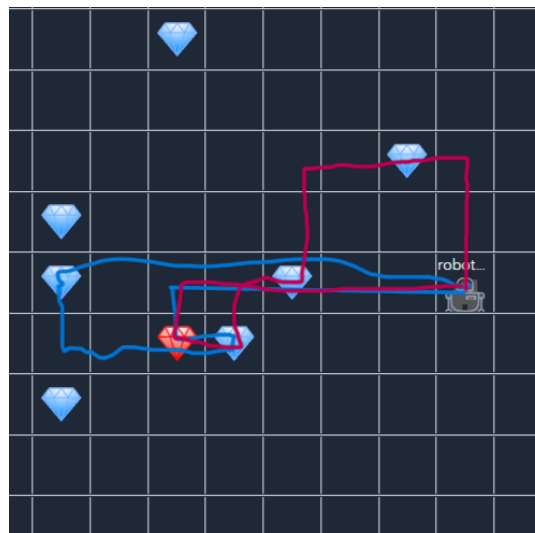


Pada kondisi ini, bot akan memilih diamond biru yang dilingkari merah. Solusi ini optimal, karena jika robot sudah memiliki 4 diamond, diamond merah tidak akan

dipilih karena tidak memenuhi syarat fungsi kelayakan sehingga akan dipilih diamond biru terdekat.



Pada kondisi ini, bot memilih jalur biru karena memprioritaskan diamond merah sehingga mengakibatkan jalur yang lebih panjang menuju base dibandingkan jalur merah. Jalur biru menghabiskan 20 langkah, sedangkan jalur merah hanya menghabiskan 16 langkah.



Pada kondisi ini, diperlihatkan juga jalur yang tidak optimal yang dipilih oleh bot karena memprioritaskan diamond merah yang jaraknya dekat.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Algoritma greedy dapat digunakan untuk mendapatkan solusi lokal optimal berdasarkan kondisi saat ini. Namun, solusi nilai optimal lokal tidak selalu sama dengan nilai optimal global. Dari pengerjaan tugas ini disimpulkan bahwa algoritma greedy dapat digunakan sebagai algoritma optimasi yang cukup baik dalam pemilihan langkah suatu bot. Hal ini disebabkan karena secara umum, solusi optimal lokal yang dihasilkan algoritma greedy akan cukup mendekati solusi optimal global atau sama dengan solusi optimal global.

5.2 Saran

Saran untuk kelompok kami adalah sebagai berikut :

1. Sebaiknya lebih mempelajari game engine dan bot engine secara mendalam untuk membantu menyelesaikan masalah yang ada pada tugas ini
2. Sebaiknya meluangkan waktu lebih banyak untuk brainstorming agar mendapatkan ide yang lebih banyak
3. Sebaiknya dapat manajemen waktu dengan lebih baik terutama dalam pengerjaan tugas ini.

LAMPIRAN

Link Github : https://github.com/RiciTrisnaP/Tubes1_Robotnik

Link Youtube : <https://youtu.be/DLBHJ8Gog44>


DAFTAR PUSTAKA

Rinaldi Munir, Diktat kuliah IF2251 Strategi Algoritmik, Teknik Informatika ITB

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag 1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag 1.pdf)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag 2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag 2.pdf)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag 3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag 3.pdf)

 Get Started with Diamonds

<https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0>

<https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1>