

**Laporan Tugas Besar 1 IF2211 Strategi Algoritma**

**Pemanfaatan Algoritma Greedy dalam Pembuatan Bot**

**Permainan Diamonds**

**Semester II Tahun 2023/2024**



**oleh**

1. Marvel Pangondian 13522075
2. Andhika Tantyo Anugrah 13522094
3. M. Hanief Fatkhan Nashrullah 13522100

**Kelompok SurtiTejo**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**

**2024**

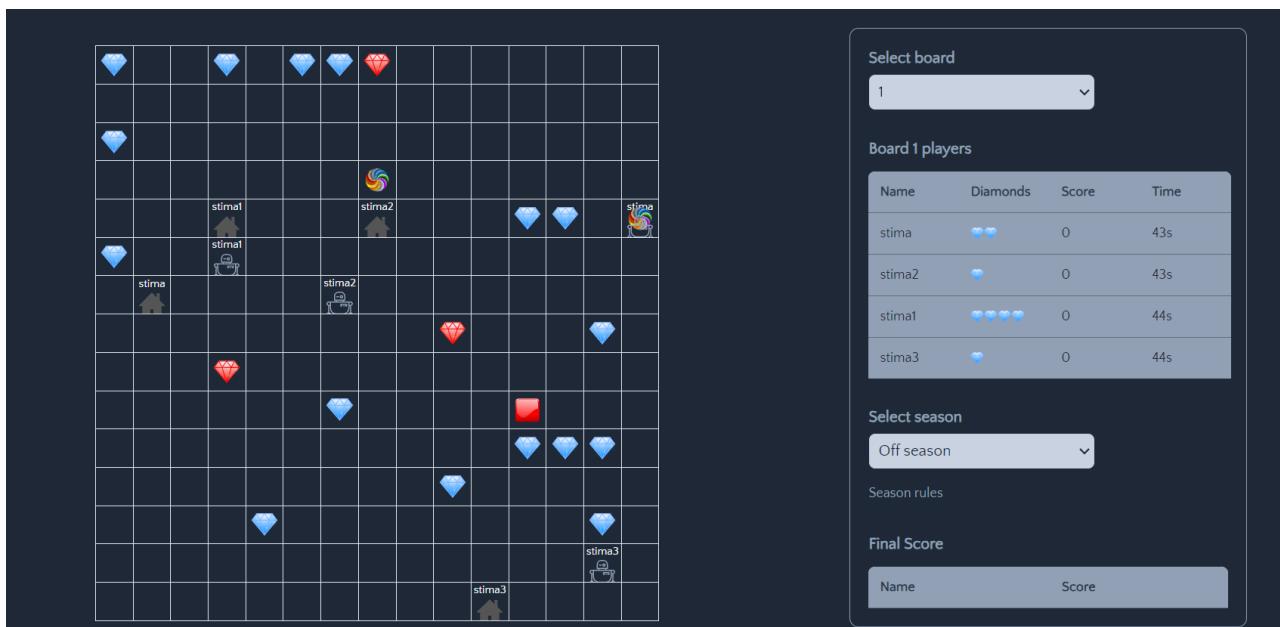
## **Daftar Isi**

Daftar Isi	2
BAB 1	
Deskripsi Persoalan	3
BAB 2	
Landasan Teori	5
2.1 Algoritma Greedy	5
2.2 Diamonds	5
BAB 3	
Aplikasi Strategi Greedy	7
3.1 Mapping Persoalan	7
3.2 Alternatif Solusi	8
3.3 Analisis Efisiensi Dan Efektivitas Solusi	9
3.4 Strategi Yang Dipilih	10
BAB 4	
Implementasi dan Uji Coba	12
4.1 Pseudocode	12
4.2 Source Code	15
4.3 Hasil Eksperimen	17
BAB 5	
Kesimpulan dan Saran	23
5.1 Kesimpulan	23
5.2 Saran	23
Lampiran	24
Daftar Pustaka	25

## BAB 1

### Deskripsi Persoalan

Diamonds merupakan suatu programming challenge yang mempertandingkan bot yang telah dibuat dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot dimana tujuan dari bot ini adalah mengumpulkan diamond sebanyak-banyaknya. Cara mengumpulkan diamond tersebut tidak 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.



Gambar 1.1.1 Contoh Permainan Diamonds

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

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



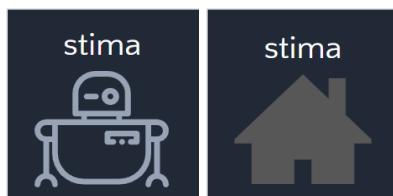
Ketika *red button* ini dilewati/dilangkahi, semua *diamond* (termasuk *red diamond*) akan di-generate kembali pada *board* dengan posisi acak. Posisi *red button* ini juga akan berubah secara acak jika *red button* ini dilangkahi.

## 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

Name	Diamonds	Score	Time
stima	♥♥	0	43s
stima2	♥	0	43s
stima1	♥♥♥♥	0	44s
stima3	♥	0	44s

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.

## BAB 2

### Landasan Teori

#### 2.1 Algoritma Greedy

Algoritma *greedy* merupakan salah satu metode yang paling sederhana untuk memecahkan persoalan optimasi. Algoritma ini selalu memilih pilihan yang terbaik yang bisa dipilih pada setiap langkah yang diambil. Pilihan terbaik ini akan memberikan solusi optimal secara lokal. Pilihan yang memberikan solusi optimal secara lokal ini diharapkan membawa menuju solusi yang optimal secara global. Algoritma *greedy* tidak selalu memberikan solusi yang optimal. Meskipun demikian, banyak masalah yang dapat diselesaikan secara optimal dengan algoritma *greedy*.

Algoritma *greedy* membentuk solusi langkah per langkah. Pada setiap langkah, terdapat lebih dari satu pilihan yang dapat dievaluasi. Dari setiap pilihan yang ada, dipilih pilihan yang memberikan hasil terbaik dari setiap langkah. Setelah langkah diambil, pilihan tidak bisa dibatalkan atau mundur ke langkah sebelumnya. Algoritma ini hanya mengambil pilihan terbaik yang ada tanpa memperhatikan konsekuensi ke depan.

Algoritma *Greedy* memiliki enam elemen, yaitu himpunan kandidat, himpunan solusi, fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi objektif. Himpunan kandidat merupakan himpunan yang berisi kandidat yang akan dipilih pada setiap langkahnya. Himpunan solusi merupakan himpunan berisi solusi yang sudah dipilih. Fungsi solusi merupakan fungsi untuk menentukan apakah himpunan kandidat yang terpilih sudah memberikan solusi atau belum. Fungsi seleksi merupakan fungsi yang memilih kandidat berdasarkan strategi *greedy* yang ditentukan. Fungsi kelayakan merupakan fungsi yang memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi. Terakhir, fungsi objektif merupakan objektif dari persoalan, yaitu mengoptimasi himpunan solusi.

#### 2.2 Diamonds

*Diamonds* adalah *programming challenge* yang disajikan dalam bentuk permainan. Setiap pemain terlebih dahulu mendesain sebuah bot yang nantinya dapat digunakan untuk bertanding melawan bot pemain lain. Tujuan dari permainan ini adalah untuk mengumpulkan berlian (*diamonds*) sebanyak mungkin dan mengantarkannya ke *base* dalam durasi waktu yang telah ditentukan. Bot yang dibuat hanya dapat menampung beberapa berlian dalam satu waktu dalam “tas”-nya. Setiap berlian

yang berhasil diantar akan memberikan bot, sebuah atau beberapa poin sesuai dengan jumlah berlian yang ada di “tas”-nya. Saat waktu berakhir, bot dengan poin terbanyak menjadi pemenang.

Hal pertama yang harus dilakukan untuk bermain adalah:

1. Mendaftarkan bot ke sebuah *endpoint* yang akan mengembalikan token rahasia yang akan digunakan untuk bermain.
2. Memeriksa papan permainan yang tersedia, lalu bergabung dengan papan yang tersedia.
3. Menjalankan bot yang sesuai dengan peraturan pada papan yang dipilih.

Menurut pengembang dari permainan *Diamonds* sendiri, para peserta boleh mendesain bot untuk mengikuti kontes dengan algoritma apapun yang peserta pikir cocok untuk memenangkan permainan, intinya, para peserta didorong untuk menghasilkan solusi yang kreatif karena permainan ini terkadang bukan hanya masalah algoritma saja. Untuk menang, diperlukan algoritma yang baik dan keberuntungan yang memadai.

## BAB 3

### Aplikasi Strategi Greedy

#### 3.1 *Mapping* Persoalan

Dalam permainan *Diamonds*, tujuan utama dari bot adalah mendapatkan poin sebanyak-banyaknya dan mendapatkan poin lebih banyak dibandingkan dengan bot kompetitor lainnya. Untuk mencapai hal tersebut, terdapat beberapa alternatif aksi yang dapat dilakukan. Contohnya, bot dapat mengambil *diamonds* lalu menyimpannya di *base*. Dapat juga dengan melakukan tackle pada bot kompetitor yang sedang membawa banyak *diamonds* lalu menyimpannya di *base*. Bisa juga dengan menggunakan portal untuk mempersingkat jarak tempuh menuju *diamonds* ataupun *base*. Beberapa aksi tersebut dan komponen lain dapat dijadikan sebagai dasar pemetaan persoalan untuk mencapai tujuan dengan menggunakan algoritma *greedy*. Berikut adalah pemetaan elemen algoritma *greedy* pada persoalan Permainan *Diamonds*.

Tabel 3.1.1. Tabel Pemetaan Persoalan

Nama Elemen	Definisi Elemen
Himpunan Kandidat	Himpunan objek yang ada pada <i>board</i> . Objek <i>diamond</i> berwarna biru memiliki 1 poin dan <i>diamond</i> merah memiliki 2 poin.
Himpunan Solusi	Objek <i>diamond</i> yang terpilih
Fungsi Solusi	Memeriksa apakah <i>inventory</i> sudah penuh dengan <i>diamond</i>
Fungsi Seleksi	Memilih <i>diamond</i> dengan poin tertinggi
Fungsi Kelayakan	Memeriksa apakah <i>diamond</i> masih cukup untuk dimasukkan ke dalam <i>inventory</i> .
Fungsi Obyektif	Langkah yang diambil untuk memenuhi <i>inventory</i> dengan <i>diamond</i> minimum.

Pemetaan tersebut masih sangat sederhana sehingga terdapat kemungkinan langkah yang diambil untuk mendapatkan *diamond* tidak minimum. Terdapat kemungkinan bot akan selalu mengambil *diamond* berwarna merah tanpa memedulikan jaraknya. Maka dari itu, pemetaan

persoalan masih dapat dikembangkan lebih lanjut. Berikut adalah pemetaan dari persoalan yang sama, tetapi juga memperhitungkan jarak dari objek target.

Tabel 3.1.2. Pengembangan dari Tabel Pemetaan Persoalan

Nama Elemen	Definisi Elemen
Himpunan Kandidat	Nilai suatu objek sebagai fungsi dari jumlah poin dan jarak objek dari bot
Himpunan Solusi	Objek yang terpilih
Fungsi Solusi	Ketika bot sudah tidak bermain lagi
Fungsi Seleksi	Memilih objek dengan nilai tertinggi
Fungsi Kelayakan	Memeriksa apakah <i>diamond</i> masih cukup untuk dimasukkan ke dalam <i>inventory</i>
Fungsi Obyektif	Jumlah nilai objek yang terpilih maksimum

## 3.2 Alternatif Solusi

### *Greedy by Distance* (Per Target)

*Greedy by distance* merupakan salah satu algoritma *greedy* yang pertama kali dipertimbangkan untuk digunakan dalam tugas besar ini. Algoritma ini bekerja dengan cara mencari objek yang terdekat dari bot. Bot akan memeriksa seluruh diamond yang ada pada board, lalu mencari diamond yang terdekat. Satu siklus dari algoritma ini adalah ketika bot telah mencapai target *diamond*. Setelah bot mencapai target, bot akan kembali ke langkah awal dan memulai siklus *greedy by distance* selanjutnya.

### *Greedy by Adjusted Point* (Per Target)

*Greedy by adjusted point* merupakan salah satu iterasi kedua algoritma *greedy* setelah diskusi lebih lanjut. Algoritma ini bekerja dengan cara mencari objek dengan poin tertinggi setelah nilai poin tersebut disesuaikan berdasarkan jarak yang dihitung dengan sebuah nilai konstanta *c*. Pada program kami, perhitungan ini dihitung dengan mengalikan poin berlian dengan konstanta *c* yang sebelumnya sudah dipangkatkan dengan jarak antara posisi bot dengan posisi berlian. Kelemahan algoritma ini adalah ketika berlian yang dikeharnya sudah diambil oleh bot lain, bot yang berjalan dengan algoritma ini tidak memperbarui status berlian tersebut dan akhirnya membuang waktu dan langkah untuk mengambil berlian yang sudah

tidak ada (*phantom diamond*). Algoritma ini sempat menjadi pilihan utama penulis tetapi setelah menemukan kelemahannya, tergantikan oleh algoritma *greedy by adjusted point (per tick)* yang lebih unggul dibandingkan algoritma ini.

### *Greedy by Adjusted Point (Per Tick)*

*Greedy by Adjusted Point* merupakan iterasi ketiga algoritma *greedy* yang ditemukan melalui *trial and error*. Algoritma ini ditemukan setelah penulis berhasil mengimplementasi algoritma *greedy by adjusted point* (per target). Pada algoritma tersebut, penulis menyadari bahwa terdapat situasi saat target menghilang dari *board*. Untuk mengatasi permasalahan tersebut, penulis berencana untuk mencari *diamond* terdekat dari bot pada setiap tick. Walaupun algoritma ini memerlukan daya komputasi yang lebih dibandingkan dengan algoritma *greedy by adjusted point* (per target), tetapi konsistensi algoritma dalam mendeteksi dan memilih *diamond* lebih dipentingkan dalam implementasi bot. Algoritma dimulai dengan mencari *diamond* berdasarkan beberapa syarat yakni jarak *diamond* serta poin *diamond*. Untuk menentukan *diamond* yang paling tepat untuk dikejar saat sebuah tick, algoritma menggunakan sebuah pengali  $c$ , jarak, serta poin sebuah *diamond*. Rumus utama dalam implementasi algoritma ini adalah  $diamond\_point * (c)^jarak\_diamond\_dari\_bot$ . *Diamond* dengan hasil paling besar berdasarkan rumus tersebut merupakan *diamond* yang akan dikejar bot. Algoritma ini dilakukan per tick untuk mencegah berlian yang sudah tidak ada (*phantom diamond*).

## 3.3 Analisis Efisiensi Dan Efektivitas Solusi

Pada permainan Etimo Diamonds, terdapat berbagai macam *state* yang dapat digunakan untuk meningkatkan kemungkinan bot menang, tetapi sebagian besar solusi algoritma yang ditemukan penulis lebih memfokuskan *state diamonds* yakni lokasi *diamonds* dan jarak *diamonds* dengan bot.

Algoritma pertama yang dikembangkan penulis adalah algoritma *Greedy by Distance* (Per Target). Algoritma ini hanya mementingkan satu hal yakni jarak target dari bot. Hasil implementasi algoritma ini memiliki kompleksitas  $O(n)$  sebab algoritma ini mengiterasi setiap target yang ada pada *board*. Dalam sisi efektivitas, algoritma ini merupakan algoritma yang cukup ringan, tetapi tidak konsisten dalam memilih target sebab algoritma ini tidak menghiraukan poin target serta *state* bot saat itu.

Algoritma kedua yang dikembangkan penulis adalah algoritma *Greedy by Adjusted Point* (Per Target). Algoritma ini akan menuju sebuah target berdasarkan jarak target dari bot serta poin yang diberikan *target* tersebut. Hasil implementasi algoritma ini memiliki kompleksitas  $O(n)$  sebab algoritma ini mengiterasi setiap target yang ada pada *board*. Algoritma ini efektif dalam memilih target paling optimal berdasarkan jarak target, poin target, serta *state* bot saat itu juga (bot akan

kembali ke *base* jika menurut algoritma tidak ada target yang pantas untuk dikejar). Algoritma ini akan terus mengincar target sampai bot berada pada lokasi target, setelah itu algoritma ini akan diulang kembali untuk mencari target berikutnya. Algoritma ini juga mengincar target berdasarkan jarak relatif target dengan portal untuk mencari *path* ke target yang paling kecil. Algoritma ini memiliki kelemahan yakni terdapat kemungkinan akan mengejar *target* yang sudah hilang sebab terdapat perubahan *state board* pada sebuah tick tertentu.

Algoritma ketiga yang dikembangkan penulis adalah algoritma *Greedy by Adjusted Point (Per tick)*. Algoritma ini mengambil *diamond* berdasarkan jarak *diamond* dari bot serta poin yang diberikan *diamond* tersebut. Hasil implementasi algoritma ini memiliki kompleksitas  $O(n)$  sebab algoritma ini mengiterasi setiap *diamond* yang ada pada *board*. Algoritma ini efektif dalam memilih *target* paling optimal berdasarkan jarak *target*, poin *target*, serta *state* bot saat itu juga (bot akan kembali ke *base* jika menurut algoritma tidak ada target yang pantas untuk dikejar). Algoritma ini juga mengincar target berdasarkan jarak relatif target dengan portal untuk mencari *path* ke target yang paling kecil. Algoritma ini akan dilaksanakan pada setiap tick untuk menghindari kejadian dimana bot mengincar target yang sudah hilang, tetapi karena itu algoritma ini memerlukan daya komputasi yang lebih besar dibandingkan dengan algoritma lainnya.

### 3.4 Strategi Yang Dipilih

Strategi algoritma *greedy* yang terpilih adalah *Greedy by Adjusted Point (Per Tick)*. Algoritma ini dipilih setelah beberapa kali iterasi dan percobaan dalam mengembangkan algoritma bot Permainan *Diamonds*. Strategi ini dipilih karena berhasil menutupi kelemahan dari dua algoritma yang diajukan sebelumnya. *Greedy by Distance* memiliki kelemahan tidak memeriksa poin dari *diamond* yang diambil. *Greedy by Distance* hanya peduli pada jarak *diamond* terdekat, lalu kembali ke *base* ketika *inventory* sudah penuh. Lalu pada *Greedy by Adjusted Point (Per Target)*, terdapat kelemahan lain yaitu *bot* dapat mengejar *diamond* yang sudah tidak ada dalam *board* lagi. Maka dari itu, strategi yang dipilih adalah *Greedy by Adjusted Point (Per Tick)*. Strategi ini akan mengambil langkah terbaik setiap kali kondisi *board* ter-update.

*Greedy by Adjusted Point (Per Tick)* bekerja dengan cara mengambil objek dengan nilai objek terbesar. Nilai objek yang dimaksud adalah nilai fungsi dari jarak dan poin dari objek. Nilai objek dihitung dengan rumus sebagai berikut:

$$f(x, y) = x \cdot k^y \quad (1)$$

Dengan  $x$  adalah poin yang dimiliki oleh objek. Objek yang dapat dipilih adalah *diamond* dan *base*. *Diamond* memiliki poin sesuai dengan jenisnya. *Diamond* merah memiliki 2 poin dan *diamond* biru memiliki 1 poin. *Base* diatur untuk memiliki jumlah poin sama dengan jumlah poin yang ada pada *inventory bot*. Variabel  $y$  adalah *manhattan distance* antara *bot* dengan objek. Variabel  $k$  adalah suatu

konstanta dengan nilai  $< 1$ . Rumus ini dipilih karena penyusutan nilai objek akan menyusut secara eksponensial seiring dengan meningkatnya *manhattan distance* dan tumbuh secara linier seiring meningkatnya poin. Hal ini mengakibatkan *bot* tidak akan memprioritaskan objek yang terlalu jauh. Rumus tersebut juga memiliki variasi lain. Variasi lain dari rumus tersebut adalah sebagai berikut:

$$f(x, y_i, y_o) = x \cdot k^{y_i + y_o} \quad (2)$$

Perbedaan antara rumus (2) dan rumus (1) adalah rumus ini menghitung *manhattan distance* untuk *teleporter*. Variabel  $y_i$  adalah *manhattan distance* antara *bot* dengan *teleporter* terdekat dari *bot* dan  $y_o$  adalah *manhattan distance* antara *teleporter* terjauh dari *bot* dengan objek di sekitar *teleporter*. Rumus ini memungkinkan untuk mendapatkan nilai lebih tinggi dibandingkan rumus (1) sehingga *bot* dapat memprioritaskan untuk masuk *entry teleporter* dan mengambil objek yang ada di sekitar *exit teleporter*.

Selain mengincar objek yang ada pada *board*, *bot* juga memperhitungkan keberadaan *bot* kompetitor. *Bot* dari kompetitor akan memiliki poin sesuai dengan jumlah poin dari *diamond* yang ada pada *inventory bot* tersebut. Untuk mengurangi kesempatan kompetitor untuk menang, *bot* akan melakukan *tackle* pada *bot* kompetitor jika *bot* memiliki *manhattan distance* 1 *block* dari *bot* kompetitor. *Bot* akan mencoba mengejar dan *tackle* sejauh 1 *block*. Jika *tackle* gagal, *bot* akan meninggalkan *bot* kompetitor dan kembali fokus pada objek yang ada pada *board*. Pengejaran sejauh 1 *block* dipilih untuk mencegah membuang langkah dan waktu yang bisa digunakan untuk mendapatkan *diamond*. Strategi ini dapat dipilih atau tidak dipilih dalam penerapan kami. Strategi ini akan dipilih jika memiliki hasil yang lebih baik dibandingkan tanpa *tackle* dalam pengujian kami.

Langkah utama dari algoritma *Greedy by Adjusted Point (Per Tick)* yang terpilih sebagai berikut:

1. Hitung seluruh nilai objek yang ada dengan rumus (1)
2. Pilih nilai objek dari rumus (1) tertinggi dan memiliki poin lebih kecil atau sama dengan sisa *slot inventory* yang tersedia
3. Hitung seluruh nilai objek yang ada dengan rumus (2)
4. Pilih nilai objek dari rumus (2) tertinggi dan memiliki poin lebih kecil atau sama dengan sisa *slot inventory* yang tersedia
5. Bandingkan nilai objek tertinggi dari kedua rumus
6. Jika nilai objek rumus (1) lebih tinggi, pilih objek dengan nilai objek tertinggi
7. Jika nilai objek rumus (2) lebih tinggi, pilih *teleporter* terdekat
8. Jika terdapat *bot* kompetitor berjarak 1 *block*, tambahkan *tackle flag* dengan 1
9. Jika *tackle flag* lebih dari 1, set menjadi 0
10. Jika *tackle flag* bernilai 0, dekati objek yang terpilih, selain itu *tackle bot* kompetitor
11. Kembali ke langkah 1

## BAB 4

### Implementasi dan Uji Coba

#### 4.1 Pseudocode

{File Imports}

{Fungsi utama yang dipanggil untuk mendapatkan langkah bot berikutnya}  
**function** next\_move(self, board\_bot: GameObject, board: Board) -> **int, int**

##### **KAMUS LOKAL**

greedyP, greedyN, greedyS : Position  
pPoint, sPoint : float  
delta\_x, delta\_y : int

##### **ALGORITMA**

{mencari lokasi diamond terbaik relatif posisi bot dan relatif posisi portal}  
greedyP, pPoint ← result\_one  
greedyN, nPoint ← result\_two

**if** (pPoint > nPoint) **then**  
    {memilih lokasi diamond relatif portal jika diamond tersebut lebih optimal}  
    greedyS ← greedyP

**else**  
    {memilih lokasi diamond relatif bot jika diamond tersebut lebih optimal}  
    greedyS ← greedyN

    {mencari langkah berdasarkan target yang dipilih}  
    delta\_x, delta\_y ← get\_direction(attribut\_posisi\_bot, attribut\_posisi\_target)

**if** (delta\_x = 0 **and** delta\_y = 0 ) **then**  
    {menghindari situasi stuck in place}  
    delta\_x, delta\_y ← get\_direction(attribut\_posisi\_bot,  
        attribut\_diamond\_pertama\_pada\_state\_board)

-> delta\_x, delta\_y

{fungsi mencari target paling optimal relatif posisi portal}  
**function** greedy(board : Board, board\_bot : GameObject) → **Position, float**

**KAMUS LOKAL**  
portals : array of position {kumpulan objek Position yang merupakan type diamond}  
maxpoint : float  
goal\_position, exitpoint, entrypoint : Position

**ALGORITMA**  
{inisialisasi goal\_position sebagai base untuk sementara}  
goal\_position ← base\_bot  
{mencari batas maksimal jarak diamond dan bot}  
  
{mencari batas maksimal jarak sebuah bot dengan diamond}  
maxpoint ← jumlah\_diamond\_pada\_bot \* (0.7) ^ jarak\_bot\_dengan\_base

```

for (i <- setiap_objek_diamond)

    {mencari jarak diamond i }
    nextpoint ←
    i.properties.points*(0.8)^SurtiTejoBot.calculate_distance(board_bot.position,i.position)

    {membandingkan jarak i dengan jarak maksimal}
    if (maxpoint < nextpoint and
        i.properties.points+board_bot.properties.diamonds<=board_bot.properties.inventory_size) then
            maxpoint <- next_point
            goal_position <- posisi_objek_i

→ goal_position, maxpoint

{fungsi mencari target paling optimal relatif posisi portal}
function greedyPortal(board : Board , board_bot : GameObject) → Position, float

KAMUS LOKAL
portals : array of position {kumpulan objek Position yang merupakan type diamond}
portal1_base, portal2_base, portal1_bot, portal2_bot, maxpoint : float
goal_position, exitpoint, entrypoint : Position

ALGORITMA
{inisialisasi portals, portal1_base, portal2_base, portal1_bot, portal2_bot}
{portal1_base dan portal2_base merupakan jarak portal - portal dengan base}
{portal1_bot dan portal2_bot merupakan jarak portal - portal dengan bot}

    {mencari portal masuk dan portal keluar}
    if (portal1_base < portal2_base) then
        exitpoint = portals[1]
    else
        exitpoint = portals[0]

    if portal1_bot > portal2_bot:
        entrypoint = portals[1]
    else:
        entrypoint = portals[0]

goal_position ← entrypoint

{mencari batas maksimal jarak sebuah bot dengan diamond}
maxpoint ← jumlah_diamond_pada_bot*(0.69)**(jarak_bot_dengan_portal_masuk +
    jarak_portal_keluar_dengan_base

for (i ← setiap_objek_diamond)

    {mencari jarak diamond i }
    nextpoint ←
    i.properties.points*(0.8)**SurtiTejoBot.calculate_distance(board_bot.position,i.position)

    {membandingkan jarak i dengan jarak maksimal}
    if (maxpoint < nextpoint and
        i.properties.points+board_bot.properties.diamonds<=board_bot.properties.inventory_size) then
            maxpoint <- next_point

```

```
goal_position <- posisi_objek_i  
→ goal_position, maxpoint
```

## 4.2 Source Code



```

● ● ●

from random import randint
import concurrent.futures
from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position

def clamp(n, smallest, largest):
    return max(smallest, min(n, largest))

def get_direction(current_x, current_y, dest_x, dest_y):
    horizontal = randint(0,1)
    delta_x = clamp(dest_x - current_x, -1, 1)
    delta_y = clamp(dest_y - current_y, -1, 1)
    if horizontal:
        if delta_x != 0:
            delta_y = 0
    else:
        if delta_y != 0:
            delta_x = 0
    return (delta_x, delta_y)

class SurtiTejoBot(BaseLogic):
    def __init__(self):
        pass

    def calculate_distance(point_a : Position, point_b : Position):
        return abs(point_b.x - point_a.x) + abs(point_b.y - point_a.y)

    def greedy(self, board, board_bot):
        goal_position = board_bot.properties.base
        maxpoint =
board_bot.properties.diamonds*(0.7)**SurtiTejoBot.calculate_distance(board_bot.position, board_bot.properties.base);
        for i in board.diamonds:
            nextpoint =
i.properties.points*(0.8)**SurtiTejoBot.calculate_distance(board_bot.position, i.position)
            if maxpoint<nextpoint and
i.properties.points+board_bot.properties.diamonds<=board_bot.properties.inventory_size:
                maxpoint = nextpoint
                goal_position = i.position
        return goal_position, maxpoint

    def greedyPortal(self, board, board_bot):
        portals = [item.position for item in board.game_objects if item.type=="TeleportGameObject"]
        portal1_base = SurtiTejoBot.calculate_distance(portals[0], board_bot.properties.base)
        portal2_base = SurtiTejoBot.calculate_distance(portals[1], board_bot.properties.base)
        portal1_bot = SurtiTejoBot.calculate_distance(portals[0], board_bot.position)
        portal2_bot = SurtiTejoBot.calculate_distance(portals[1], board_bot.position)

        if portal1_base<portal2_base:
            exitpoint = portals[1]
        else:
            exitpoint = portals[0]

```



```

if portal1_bot>portal2_bot:
    entrypoint = portals[1]
else:
    entrypoint = portals[0]

goal_position = entrypoint
maxpoint =
board_bot.properties.diamonds*(0.69)**(SurtiTejoBot.calculate_distance(board_bot.position,entrypoint)
+SurtiTejoBot.calculate_distance(exitpoint,board_bot.properties.base));
for i in board.diamonds:
    nextpoint =
i.properties.points*(0.79)**(SurtiTejoBot.calculate_distance(board_bot.position,entrypoint)
+SurtiTejoBot.calculate_distance(exitpoint,i.position))
    if maxpoint<nextpoint and
i.properties.points+board_bot.properties.diamonds<=board_bot.properties.inventory_size:
        maxpoint = nextpoint
return goal_position,maxpoint

def next_move(self, board_bot: GameObject, board: Board):
    # Get next move
    with concurrent.futures.ThreadPoolExecutor() as executor:

        future_one = executor.submit(SurtiTejoBot.greedyPortal, self, board, board_bot)
        future_second = executor.submit(SurtiTejoBot.greedy, self, board, board_bot)

        result_one = future_one.result()
        result_two = future_second.result()

        greedyP, pPoint = result_one
        greedyN , nPoint = result_two
        if (pPoint>nPoint):
            greedyS = greedyP
        else:
            greedyS = greedyN

        delta_x, delta_y = get_direction(
            board_bot.position.x,
            board_bot.position.y,
            greedyS.x,
            greedyS.y)
        # Unstuck from base
        try:
            if delta_x==0 and delta_y==0:
                print("UNSTUCK")
                delta_x,delta_y = get_direction(
                    board_bot.position.x,
                    board_bot.position.y,
                    board.diamonds[0].position.x,
                    board.diamonds[0].position.y,
                )
        except:
            pass

    return delta_x, delta_y

```

### 4.3 Hasil Eksperimen

Penjelasan :

1. SurtiTejoBot : Bot dengan Algoritma *Greedy by Adjusted Point* (Per tick)
2. BotWithTackle : Bot dengan Algoritma *Greedy by Adjusted Point* (Per tick) dengan tambahan mekanisme *tackle*.
3. OldBot : Bot dengan Algoritma *Greedy by Adjusted Point* (Per target)
4. OldBot2 : Bot dengan Algoritma *Greedy by Distance* (Per Target), nama lain dari bot ini adalah FirstSimpleSearch

Tabel 4.3.1. Tabel Perbandingan Performa antar Bot untuk sleep 1

Nama Bot	Poin Final
Round 1	
SurtiTejoBot	12
BotWithTackle	14
OldBot	3
OldBot2	6
Round 2	
SurtiTejoBot	4
BotWithTackle	11
OldBot	5
OldBot2	6
Round 3	
SurtiTejoBot	10
BotWithTackle	15
OldBot	5
OldBot2	10
Round 4	
SurtiTejoBot	13
BotWithTackle	18
OldBot	10
OldBot2	10

Round 5	
SurtiTejoBot	7
BotWithTackle	15
OldBot	10
OldBot2	7
Round 6	
SurtiTejoBot	12
BotWithTackle	6
OldBot	8
OldBot2	5
Round 7	
SurtiTejoBot	12
BotWithTackle	6
OldBot	12
OldBot2	5
Round 8	
SurtiTejoBot	9
BotWithTackle	10
OldBot	5
OldBot2	10

Tabel 4.3.2. Tabel Perbandingan Performa antar Bot untuk sleep 0.5

Nama Bot	Poin Final
Round 1	
SurtiTejoBot	17
BotWithTackle	22
OldBot	15
OldBot2	15
Round 2	

SurtiTejoBot	11
BotWithTackle	31
OldBot	19
OldBot2	13
Round 3	
SurtiTejoBot	19
BotWithTackle	9
OldBot	19
OldBot2	15
Round 4	
SurtiTejoBot	12
BotWithTackle	24
OldBot	13
OldBot2	26
Round 5	
SurtiTejoBot	20
BotWithTackle	18
OldBot	7
OldBot2	26
Round 6	
SurtiTejoBot	25
BotWithTackle	15
OldBot	16
OldBot2	15
Round 7	
SurtiTejoBot	24
BotWithTackle	20
OldBot	15
OldBot2	10

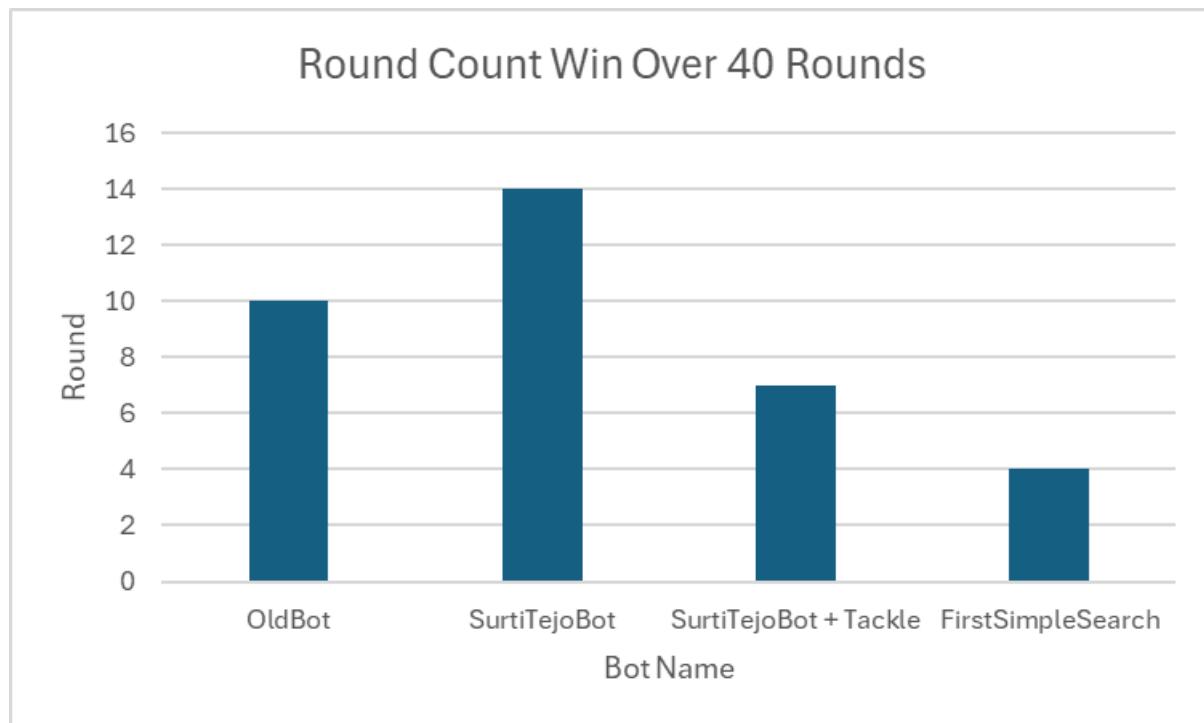
Round 8	
SurtiTejoBot	25
BotWithTackle	18
OldBot	28
OldBot2	17

Tabel 4.3.3. Tabel Perbandingan Performa antar Bot untuk sleep 0.25

Nama Bot	Poin Final
Round 1	
SurtiTejoBot	32
BotWithTackle	38
OldBot	26
OldBot2	25
Round 2	
SurtiTejoBot	50
BotWithTackle	48
OldBot	27
OldBot2	30
Round 3	
SurtiTejoBot	23
BotWithTackle	37
OldBot	43
OldBot2	27
Round 4	
SurtiTejoBot	43
BotWithTackle	13
OldBot	10
OldBot2	19
Round 5	

SurtiTejoBot	48
BotWithTackle	47
OldBot	17
OldBot2	26
Round 6	
SurtiTejoBot	47
BotWithTackle	42
OldBot	49
OldBot2	29
Round 7	
SurtiTejoBot	44
BotWithTackle	41
OldBot	29
OldBot2	44
Round 8	
SurtiTejoBot	40
BotWithTackle	42
OldBot	34
OldBot2	37

Setelah melakukan percobaan untuk setiap kondisi (sleep 1 second, 0.5 second, dan 0.25 second) penulis melakukan percobaan lain yakni 40 ronde selama 1 menit dengan sleep 1 detik. Berikut hasil percobaan tersebut



Gambar 4.3.1 Total kemenangan setiap bot tanpa menghitung ronde seri

## **BAB 5**

### **Kesimpulan dan Saran**

#### **5.1 Kesimpulan**

Melalui percobaan yang dilakukan, penulis menemukan bahwa permainan *Etimo Diamonds* dapat diselesaikan dengan algoritma *greedy*. Penulis berhasil menemukan algoritma *greedy* yang paling sesuai untuk mendapatkan *diamonds* yang cukup banyak yakni Algoritma *Greedy by Adjusted Point* (Per tick). Melalui percobaan ini, algoritma *greedy* merupakan algoritma yang cukup baik untuk menghadapi *random state* dari permainan.

Penggunaan mekanisme *tackle* tidak diterapkan pada algoritma karena dengan waktu, *delay*, dan langkah yang cukup terbatas, mekanisme *tackle* akan membuang langkah dan kesempatan untuk mendapatkan poin melalui *diamonds*. Hal tersebut terbukti dari percobaan sebanyak 40 *round* dengan *delay* 1 detik dan waktu per round 60 detik, *bot* dengan *tackle* hanya memenangkan 7 *round* sedangkan *bot* tanpa *tackle* memenangkan 14 *round*.

#### **5.2 Saran**

Berkaitan dengan masalah ini, kami memberikan beberapa saran yang perlu diperhatikan sebagai berikut:

1. Dalam algoritma *greedy* ini, penulis menyarankan untuk memprioritaskan pencarian *diamonds*.
2. Penggunaan mekanisme *tackle* ditemukan kurang efektif, berdasarkan 40 ronde yang penulis sudah lakukan ditemukan bahwa efektivitas *bot* dengan implementasi *tackle* menurun.
3. Penggunaan mekanisme *red button* dalam beberapa kasus dapat menguntungkan musuh dan merugikan diri sendiri.

## Lampiran

Link Repository : [https://github.com/hannoobz/Tubes1\\_SurtiTejo](https://github.com/hannoobz/Tubes1_SurtiTejo)

Link Video : <https://youtu.be/8dTMAh2FVZI?si=N8WVxJRcijh30NdR>

Link Hasil 40 Ronde :  Bot40Rounds.xlsx

## Daftar Pustaka

1. [Spesifikasi Tugas Besar 1 Stima 2023/2024.docx - Google Docs](#)
2. [diamonds2/RULES.md at main · Etimo/diamonds2 \(github.com\)](#)
3. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag1.pdf](#)
4. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-\(2021\)-Bag2.pdf](#)
5. [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-\(2022\)-Bag3.pdf](#)