LAPORAN TUGAS BESAR I
IF2211 STRATEGI ALGORITMA

Laporan ini dibuat untuk memenuhi tugas

Mata Kuliah IF 2211 Strategi Algortitma

Disusun Oleh:

Kelompok "Janggar"

Muhammad Fuad Nugraha (10023520)

Emery Fathan Zwageri (13522079)

Rayendra Althaf Taraka Noor (13522107)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2023/2024

Daftar Isi

BAB I
DESKRIPSI TUGAS

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

Gambar 1.1

Tampilan Permainan Diamonds

Pada tugas pertama Strategi Algoritma ini, mahasiswa diminta untuk membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentu 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**

   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 menimpaposisi 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. Dasar Teori

Algoritma Greedy merupakan algoritma yang pendekatannya *straightforward.* Stategi dari algoritam ini pun cukup sederhana, yaitu dengan melihat apa yang terlihat terbaik atau terbanyak yang bisa dilakukan saat ini, berdasarkan kondisi terkini tanpa memikirkan dampaknya pada langkah-langkah berikutnya. Algoritma ini sebagian besar digunakan pada masalah yang sederhana sehingga pendekatan *greedy* hasil bisa memberi solusi yang optimal pada permasalahan-permasalahan tersebut. Penggunaannya juga kadang digunakan pada kasus yang lebih rumit dan menghasilkan solusi yang cukup optimal, cukup baik dan cukup cepat, meskipun bukan yang paling optimal.

Pendekatan dengan algoritma greedy pada praktiknya cukup sering digunakan. Hal ini disebabkan pengimplementasiannya yang sederhana dan dapat digunakan pada sebagian besar kasus. Namun, perlu diperharikan juga bahwa solusi yang diberikannya seringkali bukan yang paling optimal. Selain itu, pendekatannya yang hanya melihat yang terbaik untuk awal penyelesaian dapat membuat proses-proses yang dilakukan selanjutnya kurang tepat dan menghasilkan hasil akhir yang jauh dari kata baiik.

2.2. Cara Kerja Program

Program permainan *Diamonds* yang dibahas pada laporan ini dijalankan dengan menggunakan 2 program, yaitu program permainan yang menjalankan papan permaian dan program bot yang akan memberi perintah untuk gerakan bot yang dimainkan di dalam *Diamonds.*

Program pertama yang menjalankan permainan akan menyiapkan sebuah papan permainan di sebuah server. Seiring berjalannya permainan, program ini akan menyediakan data-data permainan terkini yang nantinya akan diambil oleh program yang menjalankan bot.

Ketika program permainan sudah diajalankan, program bot dapat dieksekusi. Program bot akan mengambil data-data dari papan permainan, mengolah data tersebut, dan terakhir memberikan masukan ke papan permainan sebagai arah gerak bot. Proses tersebut akan terjadi berulang-ulang sepanjang berjalannya permainan. Dalam proses yang berulang inilah strategi *greedy* diterapkan. Bot-bot ini akan mengolah data yang didapatnya dari papan dan menentukan langkah kemana bot itu akan bergerak pada giliran selanjutnya. Sebagai contoh strategi *greedy,* disini bot bisa mengambil diamond terdekat darinya dan langsung membawanya ke base. Strategi lain, bot juga bisa mengumpulkan sebanyak mungkin diamond sebelum dia kembali ke base. Berbagai macam metode inilah yang menjadi tempat implementasi algoritma greedy. Setiap bot akan diatur untuk menjalankan logic (motede) tertentu dengan pendekatan greedy untuk mendapatkan diamond maksimal pada rentang waktu yang ada.

BAB III
APLIKASI STRATEGI GREEDY

3.1 Elemen-Elemen Algoritma Greedy pada persoalan Diamonds

Berdasarkan spesifikasi tugas pada bab 1, untuk mendapat diamond sebanyak mungkin elemen-elemen algoritma *greedy* pada permainan Diamonds dapat dirumuskan sebagai berikut:

- Himpunan kandidat: Berisi langkah-langkah yang dapat dilakukan yaitu ke kanan, kiri, atas, ataupun bawah
- Himpunan solusi: Himpunan diamond yang telah terambil dan dibawa ke base berdasarkan langkah-langkah yang telah diambil
- Fungsi solusi: Memeriksa apakah sekuens langkah yang diambil tidak melewati batasan papan permainan
- Fungsi seleksi: Memilih langkah menuju *diamond* yang terdekat
- Fungsi kelayakan: Memeriksa apakah *diamond* yang dituju dapat dibawa kembali ke base
- Fungsi obyektif: Memaksimalkan poin diamond yang didapat

3.2 Eksplorasi Solusi Alternatif Algoritma Greedy pada Persoalan Diamonds

Akan di bahas beberapa solusi alternatif hasil eksplorasi kami dengan algoritma greedy pada persoalan diamonds

1. Solusi utama(cintadamai.py)

Pada solusi ini strategi mencari diamond utama adalah bot mencari diamond terdekat dengan juga menghandle jarak jika lewat teleporter. Greedy ini adalah greedy pada diamond dengan Pathfinding atau pencarian jalur bot. solusi ini di desain agar tidak mudah di*tackle* oleh bot lain. Pada solusi ini juga banyak hal yang diperhatikan agar lebih efisien yang akhirnya menyebabkan kami menggunakan strategi ini sebagai strategi utama. Adapun beberapa hal yang di handle pada solusi ini sebagai berikut:

a. Pathfinding

Pada solusi ini kami memperhatikan jalur yang dilalui bot ke target. Jadi program menandai setiap posisi bot lawan yang dianggap mengancam jalur kita(jika kita jalan duluan). Jika jalannya telat dan jarak terhadap musuh yang akan ditabrak adalah ganjil tidak dihandle karena dianggap bot akan jadi terlalu lama gerak ke diamondnya.

b. Teleporter distance with diamond

Pada solusi ini kami mempertimbangkan jarak bot dengan menggunakan teleporter atau tidak. Bot akan memilih diamond dengan jarak yang lebih dekat sebagai target.

Jadi, pada solusi ini greedy nya adalah bot mencari program terdekat dengan mempertimbangkan juga jarak bot kalau lewat teleporter lalu pada perjalanannya ke target diamond bot menggunakan pathfinding untuk menghindar dari bot lawan.

2 . Solusi alternatif(begal.py)

Pada solusi ini hanya bisa diterapkan minimal dengan lawan main sebanyak 1. Tujuan utama dari greedy ini adalah mencari base lawan yang paling dekat dan potensi untuk di begal. Cara kami menentukan potensi untuk dibegal atau tidak akan kami jelaskan dibawah. Solusi ini juga bisa disebut greedy pada bot lawan.

Jadi selama eksplorasi kami menemukan bahwa jika jarak 2 bot akan tabrakan ganjil maka bot yang jalan duluan yang menang, sebaliknya jika genap yang jalan telat yang menang. Lalu karena bot harus berjalan setiap detiknya(jika tidak jalan maka akan tidak bisa jalan sampai selesai) maka menabrak bot lawan yang tidak memperhatikan pathfinding untuk menghindar adalah invariant(bisa ditentukan menang atau kalah nya). Hal ini disebabkan bahwa program yang dibuat tidak terlalu berbeda jauh kecepatannya bahkan O(N^4) saja hanya 100 ms dan pertandingan hanya 60 detik. Jadi karena bot harus tetap bergerak setiap detiknya, maka jarak bot tetap ganjil atau genap saja selamanya kecuali kalau ada teleporter yang mengubah distance dari genap menjadi ganjil atau sebaliknya. Karena tabrakan bot invarian, hal ini lah yang memotivasi kami untuk melaksanakan strategi begal. Caranya adalah bot menentukan base lawan mana yang terdekat dan sesuai syarat(Jika jarak base bot genap maka bot harus telat jalannya artinya timeleft%1000 bot kami harus lebih besar dari bot musuh,sebaliknya jika ganjil). Jika telah menemukan base lawan dengan jarak terdekat dan memenuhi syarat maka bot akan langsung menuju base lawan. Saat sampai maka bot akan mondar mandir dengan arah mendekat 1 ke arah bot lawan yang memiliki base dan balik lagi ke base lawan, sampai akhirnya bot berhasil menackle bot lawan di block(+-1,+-1) dari base lawan. Setelah selesai mentackle bot langsung balik ke base sendiri .

3. Solusi alternatif(nearest_naive.py)

Solusi ini hanya mencari diamond terdekat dengan tidak mempertimbangkan pathfinding (yolo). Bot pulang ke base jika sudah penuh inventorynya.

s

3.3 Analisis Efisiensi dan Efektifitas Solusi Alternatif Algoritma Greedy pada Persoalan Diamonds

a. cintadamai.py

Efisiensi pada program ini baik dan lumayan cepat karena kompleksitasnya O(N). Untuk efektivitas, program ini efektif karena sudah mempertimbangkan cukup aspek aspek game yang dapat membawa kepada kemenangan. Program sudah menghitung jarak dengan membandingkan jarak dengan teleport dan jarak langsung. Greedy yang digunakan juga simpel(mengambil diamond terdekat).

b. begal.py

Efisiensi pada program ini juga baik dan cepat karena kompleksitasnya O(N). Untuk efektifitas, program ini 50 50 karena tergantung jarak base dan waktu masuk permainan yang agak susah diprediksi. Program cukup efektif jika bot musuh terlalu jago dan kondisi 1 lawan 1 karena kemenangan pasti menjadi 50 50.

c. nearest_naive.py

Efesiensi program ini baik dan cepat karena kompleksitasnya O(N). Untuk efektifitas, program ini bisa dibilang jelek karena terlalu naive(tidak mempertimbangkan rintangan dan aspek aspek game lain) hanya mempertimbangkan diamond terdekat saja.

3.4 Strategi Greedy utama(cintadamai.py)

Pada akhirnya kami memilih strategi greedy cintadamai.py karena pada strategi ini paling mempertimbangkan aspek dalam game juga mempertimbangkan pathfinding di antara yang lainnya. Hal ini dikarenakan kami ingin bermain aman saja dan mengumpulkan diamond sebanyak banyaknya.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy pada Program Bot diamond

Implementasi Greedy pada program cintadamai.py

```
function next_move(self, board_bot: GameObject, board: Board):
        current_position <- board_bot.position
        # menghindari tabrakan, tombol, dan tp
        Self.halang <- []
        Teleporter <- []
        for barang in board.game_objects:
            # bot dianggap halangan jika tidak bisa ditabrak
            if  barang.type = "BotGameObject" and (barang.position /=
current_position):
                if(barang.properties.milliseconds_left and
(barang.properties.milliseconds_left mod 1000-
board_bot.properties.milliseconds_left mod 1000)>0):

self.halang.append((barang.position.x+1,barang.position.y))

self.halang.append((barang.position.x,barang.position.y+1))
                    self.halang.append((barang.position.x-
1,barang.position.y))
                    self.halang.append((barang.position.x,barang.position.y-
1))
            if barang.type = "DiamondButtonGameObject":
                self.halang.append((barang.position.x, barang.position.y))
                diabutton=barang.position
            if barang.type = "TeleportGameObject":
                print(barang)
                self.halang.append((barang.position.x, barang.position.y))
                teleporter.append((barang.position))

        props <- board_bot.properties
        self.goal_position: Optional[Position] <- None
```

```
Basepath <-
self.determinetargetndistance(current_position,teleporter,props.base)
        # kalau base sangat dekat balik
        if props.diamonds>0 and basepath[1]<=2:
            self.goal_position <- basepath[0]
        # Sisa Waktu beda tipis dengan perjalanan ke base dan masih memegang
diamond. Kembali ke base
        if props.diamonds>0 and
(board_bot.properties.milliseconds_left/1000-basepath[1])<=6:
            self.goal_position <- basepath[0]
        # diamond penuh pulang ke base
```

```
        if self.goal_position = None:
            if props.diamonds >= 4:
                self.goal_position <- board_bot.properties.base
                closestdia =
self.diamondterdekat(board,board_bot,teleporter)
                if(closestdia[1]/=100000 and closestdia[2]=1 and
props.diamonds=4): self.goal_position <- closestdia[0]
            else:

dist=self.determinetargetndistance(current_position,teleporter,diabutton)
                if(dist):
                    distlimit <- max(9,dist[1])
                else:
                    distlimit <- 100000
                closestdia <-
self.diamondterdekat(board,board_bot,teleporter)
                if(closestdia and closestdia[1]<=distlimit):
                    self.goal_position <- closestdia[0]
                else:
                    self.goal_position <- diabutton
        print (self.goal_position)
        -> self.selamatsampaitujuan(current_position,self.goal_position)

function diamondterdekat(self,board:Board, board_bot:GameObject,teleporter):
        jarak <- 100000
        for diamond in board.diamonds:
            curdia <-
self.determinetargetndistance(board_bot.position,teleporter,diamond.position
)
            if(jarak>curdia[1]):
                jarak <- curdia[1]
                target <- curdia[0]
                points <- diamond.properties.points
```

```
            # memprioritaskan diamond merah jika jaraknya tidak jauh
            if(diamond.properties.points=2 and jarak>curdia[1]*0.75):
                jarak <- curdia[1]*0.75
                target <- curdia[0]
                points <- diamond.properties.points
        -> target,jarak,points

function selamatsampaitujuan(self, curpos, target):
        dx <- target.x - curpos.x
        dy <- target.y - curpos.y
        adx <- abs(dx)
        ady <- abs(target.y - curpos.y)
```

```
        # handling bug
        if(dx = 0 and dy = 0):
            if(curpos.x+1<15):
                -> 1,0
            else: -> -1,0
        if ((adx+ady)=1):
            -> dx,dy
        else:
            if(adx>ady):
                # tidak terhalang
                if (curpos.x+(dx/adx),curpos.y) not in self.halang:
                    -> (dx/adx),0
                # terhalang
                if(ady != 0):
                    -> 0,(dy/ady)
                else:
                    # tidak di tepi
                    if(curpos.y-1 >= 0 and curpos.y+1 < 15):
                        if(curpos.x,curpos.y-1) not in self.halang:
                            -> 0,-1
                        else:
                            -> 0, 1
                    # di tepi
                    else:
                        if(curpos.y-1 >= 0):
                            -> 0,-1
                        else:
                            -> 0,1
```

```
# tidak terhalang
            if (curpos.x,curpos.y+(dy/ady)) not in self.halang:
                -> 0,(dy/ady)
            # terhalang
            if(adx != 0):
                ->-> (dx/adx),0
            else:
                # tidak di tepi
                if(curpos.x-1>=0 and curpos.x+1<15):
                    if(curpos.x-1,curpos.y) not in self.halang:
                        ->-> -1,0
```

```
                    else:
                        ->-> 1,0
                # di tepi
                else:
                    if(curpos.y-1>=0):
                        ->-> -1,0
                    else:
                        -> 1,0

    #menentukan apakah perlu teleport
    function determinetargetndistance(self,position,teleporter,target):
        t1x <- teleporter[0].x
        t1y <- teleporter[0].y
        t2x <- teleporter[1].x
        t2y <- teleporter[1].y
        p1 <- abs(position.x-target.x)+abs(position.y-target.y)
        p2 <- abs(position.x-t1x)+abs(position.y-t1y)+abs(target.x-
t1x)+abs(target.y-t2y)
        p3 <- abs(position.x-t2x)+abs(position.y-t2y)+abs(target.x-
t1x)+abs(target.x-t1y)
        if(p1<p2 and p1<p3): -> target,p1 #ga tele
        if(p2<p1 and p2<p3): -> teleporter[0],p2 #lewat teleporter 1
        -> teleporter[1],p3 #lewat teleporter 2
```

4.2 Struktur Data pada Algoritma Greedy

Struktur data pada algoritma Greedy ini diimplementasikan dengan paradigma OOP. Program terbagi menjadi beberapa modul main.py adalah program utama lalu pada folder game ada util yang isinya fungsi utility dan juga terdapat folder logic yang berisi file file logic bot. Cara implementasi logic greedy pada bot adalah dengan membuat file di folder logic dengan meng*inheritate* baselogic lalu membuat class logic baru.

Pada algoritma greedy yang diberikan, terdapat penggunaan beberapa struktur data yang berperan penting dalam menjalankan logika permainan. Berikut adalah penjelasan singkat mengenai struktur data yang digunakan:

Tabel 4.2 Struktur Data pada Algoritma Greedy

variable/method	Kegunaan
`__init__(self)`	Konstruktor class
Self.goal_positon	arah dari class
self.halang	array semua object yang menjadi rintangan
`next_move(self, board_bot: GameObject, board: Board):`	method yang menentukan tujuan bot selanjutnya
current.position	variable untuk menyimpan boardbot.position
teleporter	array untuk menyimpan teleporter

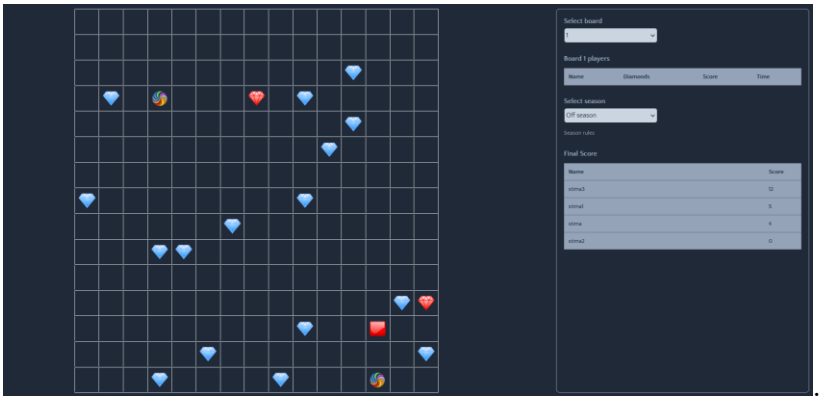
basepath	lintasan menuju base
`diamondterdekat(self,board:Board, board_bot:GameObject,teleporter):`	fungsi untuk mencari diamond terdekat
`selamatsampaitujuan(self, curpos, target):`	fungsi untuk pathfinding
`determinetargetndistance(self,position,teleporter,target):`	fungsi untuk menentukan apakah perlu teleport

4.3 Analisis Desain Solusi Algoritma Greedy

Pada solusi utama kami menggunakan greedy pada diamond, kami menghandle bot yang dapat membahayakan kami untuk menghindarinya dan potensi bot yang dapat kami tabrak tidak dianggap sebagai halangan. Berikut adalah hasil uji dari bot kami

4.3.1 Uji Permainan 4 Pemain

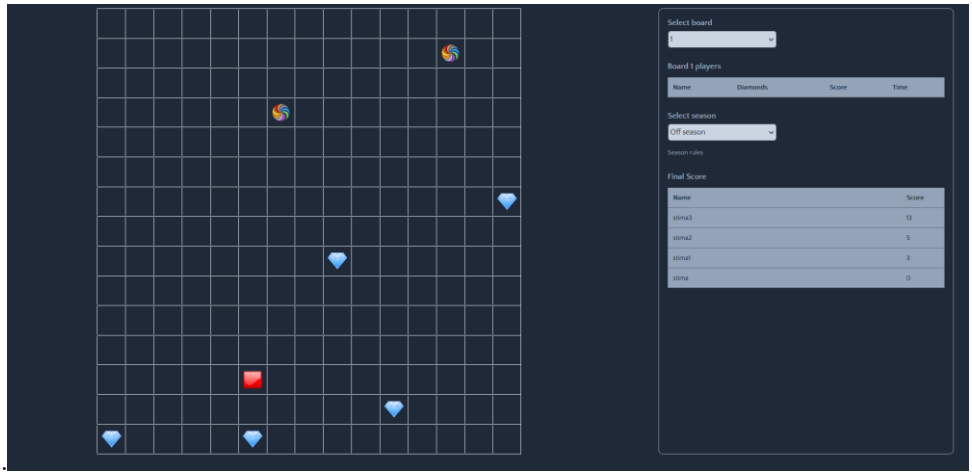
Pengujian 1:



Gambar 4.3.1.1
Hasil Permainan 4 Pemain pada Pengujian 1

Pada pengujian pertama bot stima ,stima1,stima2,stima3 merupakan bot dengan logic nearest,random,begal,damai masing masing berurutan. Dari hasil uji pertama bot dengan logic damai atau greedy by diamond and handle path menang.

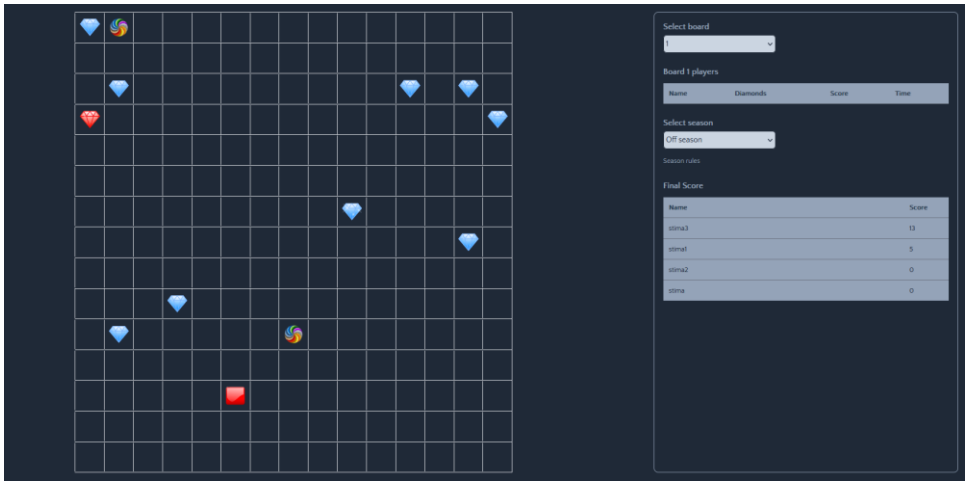
Pengujian 2:



Gambar 4.3.1.2
Hasil Permainan 4 Pemain pada Pengujian 2

Pada pengujian kedua dengan urutan nama bot dan logic yang sama .masih dimenangkan oleh bot damai

Pengujian 3:



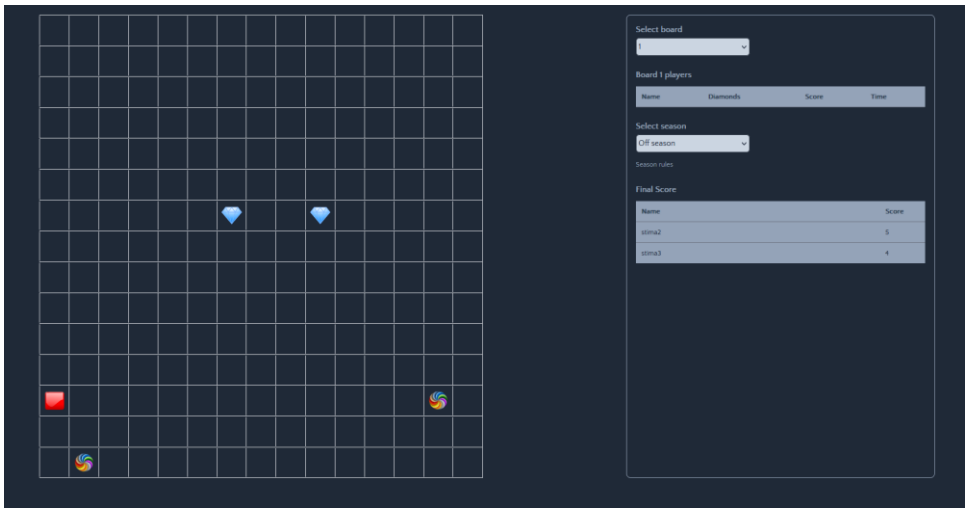
Gambar 4.3.1.3
Hasil Permainan 4 Pemain pada Pengujian 3

Pada pengujian ketiga masih dimenangkan oleh bot damai.

4.3.2.Uji Permainan 1 Lawan 1

Pada uji kali ini kami mengetest adu 1 vs 1 bot begal melawan bot damai

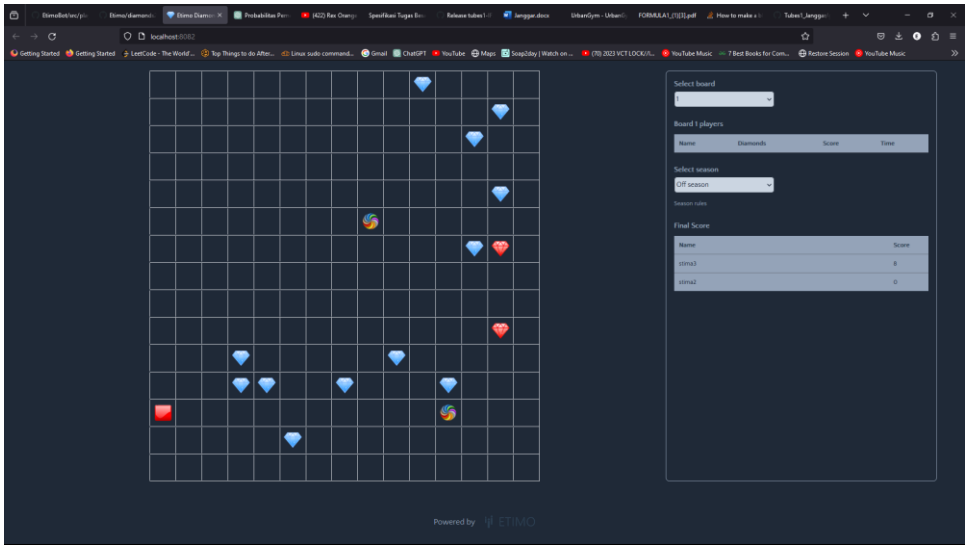
Pengujian 1:



Gambar 4.3.2.1
Hasil Permainan 1 Lawan 1 pada Pengujian 1

bot begal menang. ini terjadi karena saat mulai jarak antara dua bot ganjil dan posisinya bot begal jalna lebih dahulu.

Pengujian 2:



Gambar 4.3.2.2
Hasil Permainan 1 Lawan 1 pada Pengujian 2

hasil analisis kami pada uji uji yang telah kami lakukan pada bot damai, bot damai lebih pintar dalam menghandle jarak efektif suatu diamond dan lebih pintar dalam pathfinding. Namun ada beberapa edge case yang dapat menyebabkan bot damai kurang bisa memaksimalkan diamond yang diambil

seperti bot belum menghandle kalo bot lawan yang possible terjadi tabrakan jalan duluan sehingga distance ganjil yang menang.

BAB V
KESIMPULAN DAN SARAN

Greedy merupakan algoritma yang naif dalam menyelesaikan masalah. Maka dari itu memerlukan strategi yang tepat agar greedy dapat memecahkan masalah dengan optimal. Untuk mencapai strategi yang optimal diperlukan eksplorasi yang mendalam. Pada etimo bot kali ini greedy pada diamond dengan memperhatikan langkah dan invarian terhadap bot lawan sangat mempengaruhi kemenangan.

LAMPIRAN

Link Repository : https://github.com/mrsuiii/Tubes1_Janggar

Link Video : https://youtu.be/cUEDH3n--MM

DAFTAR PUSTAKA

Horowitz, E., Sahni, S., & Rajasekaran, S., (Tahun publikasi), Fundamentals of Computer Algorithms

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tubes1-Stima-2024.pdf