

Tugas Besar 1 IF2211 Strategi Algoritma

Semester II tahun 2023/2024

**Pemanfaatan Algoritma *Greedy* dalam Pembuatan Bot Permainan  
“Diamond”**



Disusun oleh:

Mohammad Nugraha Eka Prawira 13522001

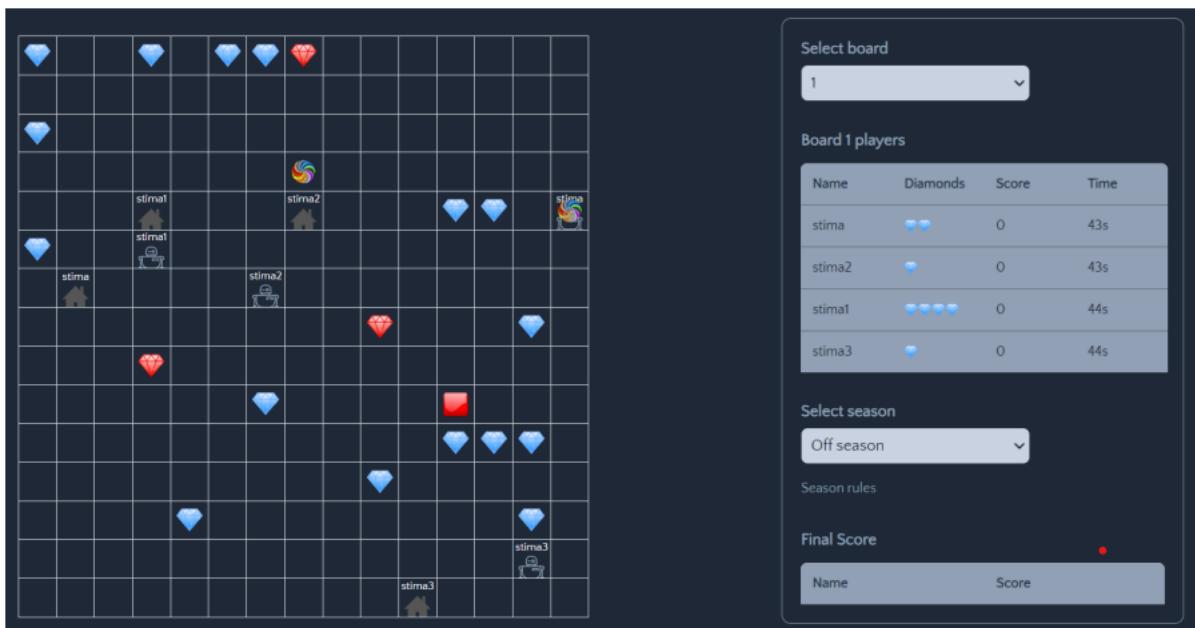
Ahmad Farid Mudrika 13522008

Muhammad Yusuf Rafi 13522009

**PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
2024**

## Bab 1: 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.



Komponen-komponen dari permainan Diamonds antara lain,

1. Diamonds:

- Terdiri dari diamond biru dan diamond merah.
- Diamond merah bernilai 2 poin, sedangkan diamond biru bernilai 1 poin.
- Diamonds di-regenerate secara berkala, dengan rasio antara diamond merah dan biru yang berubah setiap regenerasi.

2. Red Button/Diamond Button:

- Ketika dilewati, semua diamond (termasuk red diamond) di-generate kembali pada posisi acak.
- Posisi red button juga berubah secara acak setelah dilewati.

### 3. Teleporters:

- Terdapat 2 teleporter yang saling terhubung.
- Ketika bot melewati teleporter, bot akan berpindah menuju teleporter lainnya.

### 4. Bots and Bases:

- Bot digunakan untuk mengumpulkan diamond.
- Setiap bot memiliki sebuah base untuk menyimpan diamond yang dikumpulkan.
- Menyimpan diamond ke base akan menambah skor bot.

### 5. Inventory:

- Tempat penyimpanan sementara diamond yang telah diambil oleh bot.
- Memiliki kapasitas maksimum.
- Bot dapat menyimpan isi inventory ke base agar inventory kosong kembali.

Adapun, komponen 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.
  - c. Program utama (main) dan utilitas lainnya.

Pada tugas besar pertama Strategi Algoritma ini, kami mengimplementasikan algoritma *Greedy* pada bot permainan Diamonds dalam bahasa Python dengan tujuan memenangkan permainan membuat sebuah bot yang nantinya akan dipertandingkan satu sama lain. Tentunya, perubahan algoritma yang dilakukan hanya pada program bot logic.

## Bab 2: Landasan Teori

### 2.1 Algoritma Greedy secara Umum

Algoritma greedy adalah pendekatan yang mengaplikasikan prinsip "greedy" dalam penyelesaian masalah optimasi, dengan tujuan memaksimalkan atau meminimalkan suatu parameter tertentu. Dalam algoritma ini, masalah dibagi menjadi langkah-langkah kecil, dan solusi dibentuk secara bertahap. Pada setiap langkah, pemrogram harus membuat keputusan terbaik berdasarkan informasi yang tersedia saat itu, sering disebut sebagai "greedy choice". Namun, backtracking tidak diizinkan dalam algoritma greedy, sehingga pemrogram harus memilih solusi yang optimal secara lokal pada setiap langkahnya.

Dalam algoritma greedy, terdapat beberapa elemen yang perlu didefinisikan:

1. Himpunan kandidat ( $C$ ): Berisi kandidat yang dapat dipilih pada setiap langkah.
2. Himpunan solusi ( $S$ ): Berisi kandidat yang telah dipilih sebagai bagian dari solusi.
3. Fungsi solusi (*solution function*): Menentukan apakah himpunan solusi yang dikumpulkan sudah memberikan solusi sesuai yang diharapkan.
4. Fungsi seleksi (*selection function*): Fungsi ini bersifat heuristik dan bertujuan untuk memilih kandidat berdasarkan strategi *greedy* tertentu.
5. Fungsi kelayakan (*feasibility function*): Memeriksa apakah kandidat yang terpilih oleh fungsi seleksi dapat dimasukkan ke dalam himpunan solusi.
6. Fungsi objektif (*objective function*): Memaksimumkan atau meminimumkan suatu parameter pada suatu persoalan. (Domain: himpunan objek, Range: himpunan objek).

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa:

"Algoritma *greedy* melibatkan pencarian sebuah himpunan bagian,  $S$ , dari himpunan kandidat,  $C$ ; yang dalam hal ini,  $S$  harus memenuhi beberapa kriteria yang ditentukan, yaitu  $S$  menyatakan suatu solusi dan  $S$  dioptimisasi oleh fungsi obyektif".

Berikut Skema umum algoritma *greedy* menggunakan *pseudocode*:

```

function greedy(C : himpunan_kandidat) → himpunan_solus
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }

Deklarasi
    x : kandidat
    S : himpunan_solus

Algoritma:
    S ← {} {inisialisasi S dengan kosong}
    while (not SOLUSI(S)) and (C ≠ {}) do
        x ← SELEKSI(C) {pilih sebuah kandidat dari C}
        C ← C – {x} {buang x dari C karena sudah dipilih}
        if LAYAK(S ∪ {x}) then {x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi}
            S ← S ∪ {x} {masukkan x ke dalam himpunan solusi}
        endif
    endwhile
{SOLUSI(S) or C = {}}

if SOLUSI(S) then {solusi sudah lengkap}
    return S
else
    write('tidak ada solusi')
endif

```

Gambar 2.1. *Pseudocode* algoritma *greedy*

Kesimpulannya, algoritma greedy cocok digunakan untuk masalah yang memerlukan solusi yang cukup akurat dan tidak memerlukan solusi yang mutlak optimal. Solusi yang diberikan oleh algoritma greedy lebih efisien dibandingkan dengan algoritma yang mencari solusi eksak namun memerlukan waktu komputasi yang besar.

## 2.2 Garis Besar Cara Kerja Bot Permainan Diamonds

Cara kerja bot untuk permainan Diamonds ini berada pada Bot starter pack di script main.py. Berikut adalah fungsi-fungsi penting dalam pemrograman utama:

1. **\*\*Api\*\***: Kelas untuk berinteraksi dengan API permainan.
  - **\*\*api.list\_boards()\*\***: Mengambil daftar papan permainan yang tersedia.
2. **\*\*BoardHandler\*\***: Kelas untuk menangani data papan permainan.
  - **\*\*board\_handler.get\_board(board\_id)\*\***: Mendapatkan informasi tentang papan permainan yang sedang berlangsung menggunakan ID papan permainan yang diberikan.
3. **\*\*BotHandler\*\***: Kelas untuk menangani informasi bot.

- **\*\*bot\_handler.recover(email, password)\*\*:** Mencoba untuk memulihkan token bot menggunakan email dan password yang diberikan.
  - **\*\*bot\_handler.register(name, email, password, team)\*\*:** Mendaftarkan bot baru dengan nama, email, password, dan tim yang diberikan.
  - **\*\*bot\_handler.get\_my\_info(token)\*\*:** Mendapatkan informasi tentang bot yang sedang berjalan menggunakan token yang diberikan.
  - **\*\*bot\_handler.join(bot\_id, board\_id)\*\*:** Bergabung dengan papan permainan menggunakan ID bot dan ID papan permainan yang diberikan.
  - **\*\*bot\_logic.next\_move(bot, board)\*\*:** Menghitung langkah berikutnya yang harus diambil oleh bot berdasarkan logika yang telah ditentukan.
  - **\*\*bot\_handler.move(bot\_id, board\_id, delta\_x, delta\_y)\*\*:** Melakukan langkah pada papan permainan dengan menggerakkan bot ke posisi yang ditentukan.
4. **\*\*RandomLogic\*\*, \*\*BaseLogic\*\*, \*\*BotsMove\*\*, \*\*kangTackle\*\*:** Logika untuk menggerakkan bot dalam permainan.
  5. **\*\*sleep\*\*:** Fungsi untuk memberikan jeda dalam eksekusi program.

Adapun, Garis besar cara kerja bot dalam permainan Diamonds adalah sebagai berikut:

1. **\*\*Mengambil Token atau Registrasi Bot\*\*:** Bot mencoba untuk mendapatkan token atau melakukan registrasi jika tidak ada token yang diberikan.
2. **\*\*Setup Bot\*\*:** Bot memperoleh informasi tentang logika kontrol yang akan digunakan dan mendapatkan informasi tentang bot yang sedang berjalan.
3. **\*\*Mencari Papan Permainan\*\*:** Bot mencoba untuk bergabung dengan papan permainan yang tersedia.
4. **\*\*Persiapan State dari Papan Permainan\*\*:** Bot mempersiapkan state dari papan permainan untuk dimainkan.
5. **\*\*Loop Permainan\*\*:** Bot berada dalam loop permainan, di mana bot terus melakukan langkah-langkah berikut:
6. **\*\*Menghitung Langkah Berikutnya\*\*:** Bot menggunakan logika yang telah ditentukan untuk menghitung langkah berikutnya.

7. **\*\*Melakukan Validasi Langkah\*\*:** Bot memvalidasi langkah yang dihitung untuk memastikan bahwa langkah tersebut valid.
8. **\*\*Melakukan Langkah\*\*:** Bot mencoba untuk melakukan langkah yang telah dihitung.
9. **\*\*Memperbarui State\*\*:** Bot memperbarui state dari papan permainan setelah melakukan langkah.
10. **\*\*Memberikan Jeda\*\*:** Bot memberikan jeda sebelum melakukan langkah berikutnya.
11. **\*\*Game Over\*\*:** Permainan berakhir ketika bot tidak lagi dapat melakukan langkah yang valid.

Dengan demikian, bot diimplementasikan untuk berinteraksi dengan permainan Diamonds, menggunakan logika kontrol yang telah ditentukan untuk membuat keputusan tentang langkah yang harus diambil dalam permainan.

### **2.3. Implementasi Algoritma Greedy ke dalam Bot Permainan Diamond**

Inti dari program terkait cara kerja bot adalah logic dari bot itu sendiri, yakni dilakukan pada pemanggilan fungsi **bot\_logic.next\_move(bot, board)**. Maka dari itu, semua rencana strategi algoritma *greedy* akan diimplementasikan pada fungsi tersebut.

Penggunaan algoritma *greedy* sebagai algoritma pembentukan *bot* sangat tepat dikarenakan tidak memerlukan waktu atau *resource* yang terlalu banyak karena dalam permainan ini sendiri terdapat batasan waktu. Meskipun terdapat beragam pilihan solusi *greedy* yang tersedia bagi penulis, mereka mungkin tidak selalu dapat menghasilkan solusi global yang optimal. Akan tetapi, penulis dapat tetap mencapai solusi lokal yang optimal, meskipun nilainya mungkin tidak sama dengan solusi global yang optimal.

## Bab 3: Aplikasi Strategi *Greedy*

### 3.1. Pemetaan Elemen/Komponen Algoritma *Greedy* pada Bot Permainan Diamond

Dalam permainan Diamond, tujuan setiap *bot* pemain berusaha untuk mengumpulkan skor terbanyak. Terdapat banyak cara untuk meraih hal tersebut, seperti berusaha mencuri *diamond* dari *bot* lawan, mencari *diamond* terdekat, dan lain-lain.

| Nama Elemen/Komponen | Definisi Elemen/Komponen   |
|----------------------|--|
| Himpunan kandidat    | Seluruh kotak yang terdapat pada <i>board</i> permainan Diamond.   |
| Himpunan solusi      | Himpunan kotak-kotak yang ditentukan sebagai posisi <i>goal</i> .  |
| Fungsi solusi        | Memeriksa apakah kotak yang dipilih merupakan elemen terakhir dari himpunan solusi saat ini.                       |
| Fungsi seleksi       | Memilih kotak yang dinilai paling optimal sesuai strategi greedy yang dipilih                                      |
| Fungsi kelayakan     | Memeriksa apakah koordinat kotak yang dipilih ada di dalam <i>board</i> .  |
| Fungsi objektif      | Mencari urutan kotak-kotak <i>goal</i> yang membuat <i>bot</i> pemain memenangkan permainan dengan skor tertinggi. |

### 3.2. Eksplorasi Alternatif Solusi Algoritma *Greedy* pada Bot Permainan Overdrive

Terdapat banyak sekali alternatif solusi algoritma *greedy* yang dapat diimplementasikan pada *bot* permainan Diamond. Hal ini dikarenakan terdapat banyak elemen dari *game engine* yang dapat diakses oleh *bot* dan kemudian diubah *state* pada elemen tersebut. Selain itu, elemen-elemen tersebut saling berinteraksi dengan elemen lainnya sehingga jika terdapat perubahan pada suatu *state* elemen, terdapat peluang bahwa *state* lainnya berubah pula. Sebagai contoh, elemen *red button* pada *board* dapat me-reset dan mengacak posisi semua objek di *board*. Oleh karena itu, strategi algoritma *greedy* yang disusun oleh penulis memiliki interaksi dengan strategi algoritma *greedy* pada bidang yang berbeda. Berikut ini merupakan beberapa alternatif strategi algoritma *greedy* yang dapat diimplementasikan pada *bot* pemain:

#### 3.2.1. Strategi Greedy pada *bot* lawan

Seperti yang telah diketahui sebelumnya, dalam suatu permainan Diamond, umumnya terdapat 4 *bot* yang berpartisipasi dalam satu waktu. Tiap *bot* akan mengimplementasikan strateginya masing-masing untuk mendapat skor tertinggi. Tiap bot memiliki inventori yang dapat menampung maksimal 5 diamond. Umumnya, ketika inventori penuh, bot akan kembali ke base.

Strategi yang dapat diimplementasikan memanfaatkan mekanisme *tackle*, yaitu keadaan dimana dua bot berada di kotak yang sama. Dalam keadaan ini, *bot* pertama akan dikembalikan ke basenya dengan inventori kosong, dan *bot* kedua akan mengambil semua *diamond* dalam inventori *bot* pertama selama inventornya belum penuh. Jadi, strategi yang digunakan adalah untuk mengejar *bot* musuh untuk melakukan *tackle* dan mengambil *diamond*-nya. Apabila inventori *bot* sudah penuh, *bot* akan kembali ke base.

### 3.2.2. Strategi Greedy *diamond*

Dalam suatu permainan *diamond*, tujuan utama tiap pemain adalah untuk mengumpulkan *diamond* di inventori sebanyak dan secepat mungkin, lalu kembali ke *base* untuk “menyetorkan” *diamond* di inventori. Umumnya, waktu tiap permainan adalah 60 detik. Terdapat 2 jenis *diamond* di board, yaitu *diamond* merah dan *diamond* biru, dengan *diamond* merah bernilai 2 poin.

Strategi ini adalah strategi paling sederhana dan solid, yaitu hanya fokus ke *diamond* terdekat. Dalam tiap gerakan, jika *bot* belum memiliki lokasi tujuan spesifik dan inventori belum penuh, *bot* akan membaca posisi semua *diamond* di map, termasuk *diamond* merah, mengurutkannya berdasarkan jarak, lalu memilih *diamond* terdekat sebagai lokasi tujuan. Jika inventori sudah penuh, *bot* akan memilih *base* sebagai lokasi tujuan.

## 3.3. Analisis Efisiensi dari Kumpulan Solusi Algoritma *Greedy*

Pada permainan Diamond, banyak *gamestate* yang bisa kita ketahui dengan mudah seperti posisi pemain, posisi lawan, inventori pemain, inventori lawan, dan lain-lain. Tentu hal tersebut memudahkan program agar berjalan dengan efektif. Pemain dapat mendeteksi keseluruhan map sehingga pemain dapat merencanakan pilihan paling optimal.

Semua objek pada *board* dapat dilihat pada atribut *game\_object* dari objek *board*. Hal ini mengakibatkan semua pencarian objek, baik yang paling sederhana hingga yang paling rumit memiliki kompleksitas waktu tetap  $m \times n$ , dengan  $m$  adalah jumlah baris di

kotak dan n jumlah kolom. Dalam hal ini, kita dapat menganggapnya O(1).

Pada strategi Greedy *bot* lawan, kita melakukan pencarian posisi *bot* lawan secara iteratif, lalu menghitung jarak tiap *bot* lain dari *bot* kita, mengurutkannya, dan menjadikan *bot* terdekat sebagai tujuan. Karena ada pengurutan di dalamnya, kompleksitas waktunya O( $n \log n$ ).

Pada strategi Greedy *diamond*, kita melakukan pencarian posisi semua *diamond* di *board*. Meskipun terdapat *method* diamonds pada *board*, *method* ini tetap melakukan pencarian iteratif, sehingga kompleksitas waktunya sama dengan strategi lain. Setelah mendapat list *diamond* di *board* dan posisinya, kita menghitung jarak tiap *diamond* dari kita, lalu mengurutkannya dari yang terkecil. Kita lalu memilih *diamond* terdekat sebagai tujuan kita.

### 3.4. Analisis Efektivitas dari Kumpulan Solusi Algoritma *Greedy*

#### 3.4.1. Efektivitas Strategi Greedy pada *Bot* Lawan

Strategi ini belum tentu berdampak baik terhadap performa *overall* dari *bot*. Hal ini dikarenakan luasnya *board* dan sedikitnya waktu yang diberikan. Umumnya, satu permainan Diamond memiliki batas waktu 60 detik, dengan waktu *sleep* 1 detik setelah tiap gerakan. Pada saat *bot* dalam keadaan *sleep*, *bot* tidak dapat melakukan perhitungan untuk gerakan selanjutnya. Ini berarti dalam satu permainan diamond, gerakan yang dilakukan tiap *bot* umumnya kurang dari 60.

Dengan jumlah kotak dalam suatu *board* umumnya 225 (15x15), strategi ini akan membutuhkan waktu yang lama untuk mendekati *bot* lawan. Bahkan jika *bot* lawan ada di dekat *bot* kita, *bot* lawan mungkin mengimplementasikan strategi untuk menghindari *bot* kita. Bahkan jika *bot* lawan tidak mengimplementasikan strategi untuk menghindar, karena batasan waktu *sleep*. Seberapa cepat pun *bot* kita, karena batasan waktu *sleep*, dan *bot* lawan tidak mungkin diam di tempat, kita membutuhkan waktu yang lama untuk mencapai *bot* lawan.

#### 3.4.2. Efektivitas Strategi Greedy pada *Diamond*

Strategi ini belum tentu optimal, jika *diamond* terdekat ada di arah yang berlawanan dengan beberapa *diamond* terdekat setelahnya. Contoh, jika *bot* saat

ini ada di koordinat (1,6), *diamond* terdekat ada di koordinat (1,1), *diamond* terdekat kedua ada di (1,14), dan *diamond* terdekat ketiga, keempat, kelima, dst. ada di dekat *diamond* kedua, pilihan untuk menuju ke (1,1) tidaklah optimal. Selain itu, strategi ini tidak menggunakan objek-objek lain di *board*.

### 3.5. Strategi *Greedy* yang Digunakan pada Program *Bot*

Strategi yang penulis ambil sebagai algoritma *greedy* utama dari program *bot* permainan Overdrive ini adalah peningkatan dari strategi yang dipaparkan pada subbab 3.2. Penulis mengambil seluruh strategi lalu kekurangan-kekurangan yang dipaparkan di subbab 3.4.2. Agar seluruh strategi heuristik dapat digabungkan menjadi suatu algoritma *greedy*, penulis harus mendefinisikan urutan dari objek-objek mana yang harus diincar terlebih dahulu. Salah satu cara mendefinisikan urutan tersebut adalah dilihat dari seberapa besar prioritas suatu objek jika dibandingkan dengan objek lainnya. Pemrogram dapat menggunakan logika atau mengecek urutan objek untuk memformulasikan urutan mana saja yang dapat diimplementasikan sebagai algoritma *greedy*.

Setelah dipikirkan dan kemudian mencobanya pada *game engine*, penulis telah berhasil memformulasikan strategi algoritma *greedy* yang penulis rasa paling optimum. Strategi Greedy yang penulis pilih melibatkan memilih kotak terdekat dari zona(kotak 5x5) yang memiliki diamond terbanyak di dalamnya jika inventori belum penuh. Apabila inventori *bot* sudah penuh, pilih base. Apabila *bot* sudah mencapai zona tujuan, pilih diamond terdekat. Ulangi hingga *diamond* di zona habis atau inventori penuh. Apabila zona yang mengandung bot tidak memiliki diamond dan memiliki red button, pilih red button. Dalam menuju ke kotak yang diinginkan, bandingkan jarak jika menggunakan teleporter dan tidak. Pilih yang terdekat. Dalam menuju ke kotak tujuan, apabila kotak tujuan saat ini adalah *base* dan terdapat *bot* lain dalam zona yang sama, pilih kotak dengan formula (*tujuan* = *tujuan\_awal*+*tujuan\_awal*-*posisi\_bot\_lain*). Apabila koordinat hasil formula tidak ada di *board*, pilih kotak terdekat di *board* dari koordinat hasil formula.

Penulis tidak mengimplementasikan strategi untuk men-*tackle* *bot* lawan karena sesuai dengan paparan di subbab 3.4.1, strategi untuk mengejar *bot* lawan dinilai kurang efektif.

## Bab 4: Implementasi dan Pengujian

### 4.1. Implementasi Algoritma *Greedy* pada Bot Permainan Diamond

Implementasi algoritma greedy pada program terdapat pada file BotsMove.py, didalamnya terdapat algoritma next\_move yang sudah disesuaikan dengan algoritma greedy sesuai kriteria yang ditentukan, serta beberapa fungsi lain yang membantu.

Berikut adalah pseudocode dari kelas BotsMove :

```

ALGORITMA:
procedure __init__(self)
    self.directions ← [(1, 0), (0, 1), (-1, 0), (0, -1)]
    self.block in ← [(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)]
    self.goal position: Optional[Position] ← None
end procedure

function boundary(self, n, smallest, largest) → Integer
    →max(smallest, min(n, largest))
end function

function get_way(self, current_x, current_y, dest_x, dest_y, base, telestart:Position,
teletarget:Position, checktele) →(Integer,Integer)
    delta_x ← self.boundary(dest_x - current_x, -1, 1)
    delta_y ← self.boundary(dest_y - current_y, -1, 1)
    check ← False
    if ¬ checktele then
        next_pos_x ← current_x + delta_x
        next_pos_y ← current_y + delta_y
        if (next_pos_x = telestart.x ∧ next_pos_y = telestart.y) then
            if (next_pos_x = telestart.x ∧ current_y+1 = telestart.y) ∧ (next_pos_y
            ≠ telestart.x) then
                delta_y ← 0
            else
                delta_x ← 0
            end if
            check←True
        else if (next_pos_x = teletarget.x ∧ next_pos_y =
        teletarget.y) then
            if (next_pos_x = teletarget.x ∧ current_y+1 = teletarget.y) ∧
            (next_pos_y≠ teletarget.x) then
                delta_y ← 0
            else
                delta_x ← 0
            end if
            check ← True
        
```

```

end if

end if

if (delta_x=0 ∧ delta_y=0) then
    if (current_x = 0 ∨ current_y = 0) then
        delta_x ← 1
    else if (current_x = 14 ∨ current_y = 14) then
        delta_x ← -1
    else
        delta_x ← 1
        delta_y ← 0
    end if
else if ¬ check then
    if delta_x ≠ 0 then
        delta_y ← 0
    end if
end if
→(delta_x, delta_y)
end function

procedure chase(self, bot2:GameObject)
    self.goal position ← bot2.position
end procedure

function get_tacklebot(self, board:Board, Board_bot)→list of GameObjects
    item ← board.game objects
    tacklers ← []
    for a ∈ item do
        if a.type="BotGameObject" ∧ a.properties.can_tackle ∧ a.position
        /=Board_bot.position then
            tacklers.append(a)
        end if
    end for
    next pos y
    →tacklers
end function

function get_rebut_telep(self, board, Board_bot)→(Position, List)
    item board ← board.game objects
    teleports ← []
    current_position ← Board_bot.position
    checkred ← False
    checktele ← False
    for item ∈ item board do
        if checkred ∧ checktele then
            break
        else
            if item.type='DiamondButtonGameObject' then
                red ← item
                checkred ← True
            else if item.type='TeleportGameObject' then

```

```

teleports.append((abs(abs(item.position.x - current_position.x)
+ abs(item.position.y - current_position.y)), item.position))
if len(teleports) > 1 then
    sorted_teleport ← sorted(teleports, key ← lambda x:
        x[0])
    checktele ← True
end if
end if
end for
redpos ← red.position
→redpos, sorted_teleport
end function

function current_totalpointblock(self, current_position, diamond_objects) → (Integer, List)
    whichBlockx ← current_position.x//5
    whichBlocky ← current_position.y//5
    block_start_row ← whichBlockx * 5
    block_end_row ← (whichBlockx + 1) * 5
    block_start_col ← whichBlocky * 5
    block_end_col ← (whichBlocky + 1) * 5
    totalpointblock ← 0
    listrangediamond ← []
    for k ∈ range(block_start_row, block_end_row) do
        for l ∈ range(block_start_col, block_end_col) do
            for diamond ∈ diamond_objects do
                if diamond.position.x=k ∧ diamond.position.y=l then
                    totalpointblock+ ← diamond.properties.points
                    listrangediamond.append((abs(abs(diamond.position.x
                        - current_position.x) + abs(diamond.position.y -
                        current_position.y)), diamond.position,
                        diamond.properties.points))
                end if
            end for
        end for
    end for
    →totalpointblock, listrangediamond
end function

function totalpointblock(self, start_position, diamond objects) → (Position)
    totaleveryblock ← []
    for i ∈ range(3) do
        for j ∈ range(3) do
            block_start_row ← 5 * i
            block_end_row ← (1 + i) * 5
            block_start_col ← j * 5
            block_end_col ← (1 + j) * 5
            totalpointblock ← 0
            listrangediamond ← []
            for k ∈ range(block_start_row, block_end_row) do
                for l ∈ range(block_start_col, block_end_col) do

```

```

for diamond ∈ diamond_objects do
    if diamond.position.x=k ∧
        diamond.position.y=l then
            totalpointblock+ ←
                diamond.properties.points
            listrangediamond.append((abs(abs(dia-
                mond.position.x - start_position.x) +
                abs(diamond.position.y -
                start_position.y)), diamond.position))
        end if
    end for
end for
sorted_listrangediamond ← sorted(listrangediamond, key← lambda x:
    x[0])
if sorted_listrangediamond then
    distance, locationdiamond ← sorted listrangediamond[0]
    totaleveryblock.append((totalpointblock, distance,
        locationdiamond))
end if
end for
end for
sorted_totaleveryblock ← sorted(totaleveryblock, key← lambda x: x[1])
→, self.goal position ← sorted_totaleveryblock[0]

→self.goal position
end function

function next move(self, Board_bot: GameObject, board: Board) → (Integer,Integer)
    props ← Board_bot.properties
    base ← Board_bot.properties.base
    current_position ← Board_bot.position
    distanceToBase ← abs(abs(base.x- current_position.x) + abs(base.y -
        current_position.y))
    redpos, teleport ← self.get_rebut_telep(board, Board_bot)
    →telestart ← teleport[0]
    →teletarget ← teleport[1]
    checktele ← False

    if props.diamonds = 5 ∨ props.diamonds = 4 ∨ ((distanceToBase + 1 =
        (props.milliseconds left/1000)) ∧ props.diamonds ≠ 0) ∨ (distanceToBase =
        (props.milliseconds left/1000)) then
        base ← Board_bot.properties.base
        self.goal position ← base
        bots ← self.get tacklebot(board, Board_bot)
        for bot ∈ bots do
            if (bot.position.x//5 = current_position.x//5) ∧ bot.position.y//5=current
            then
                delx ← self.goal position.x-bot.position.x
                dely ← self.goal position.y-bot.position.y
                self.goal position.x+ ← delx
                self.goal position.y+ ← dely

```

```

        if self.goal position.x>14 then
            self.goal position.x ← 14
        end if
        if self.goal position.x<0 then
            self.goal position.x ← 0
        end if
        if self.goal position.y>14 then
            self.goal position.y ← 14
        end if
        if self.goal position.y<0 then
            self.goal position.y ← 0
        end if
        break
    end if
end for
else
    diamond_objects ← board.diamonds
    totalpointblock, listrangediamond ←
    self.current_totalpointblock(current_position,diamond_objects)
    _,total_teletarget,_ ← self.current_totalpointblock(teletarget, diamond_objects)

    if totalpointblock=0 then

        if ((redpos.x//5 = current_position.x//5) ∧ (redpos.y//5=
        current_position.y//5)) then
            self.goal position ← redpos
        else if ((telestart.x//5 = current_position.x//5) ∧ (telestart.y//5=
        current_position.y//5)) ∧ (total teletarget≠ 0) then
            checktele ← True
            self.goal position ← telestart
        else
            self.goal_position ← self.totalpointblock(current_position,
            diamond_objects)
        end if
    else
        sorted_listrangediamond ← sorted(listrangediamond, key ← lambda x:
        x[0])
        if props.diamonds=4 ∧ sorted_listrangediamond[0][2]=2 then
            sorted_listrangediamond.pop(0)
        end if
        if len(sorted_listrangediamond)=0 then
            self.goal_position ← base
        else
            _, self.goal position, ← sorted_listrangediamond[0]
        end if
    end if
    delta_x, delta_y ← self.get way(currentposition.x, currentposition.y, self.goal
    position.x,self.goal position.y,base,telestart,teletarget,checktele)

    →delta_x, delta_y
end function

```

## 4.2. Penjelasan Struktur Data pada Bot Permainan Diamonds

Struktur data pada permainan Diamonds ini berbentuk *class*. Berikut pemaparan lebih mendalam mengenai beberapa class yang ada pada *bot* Diamonds:

### 1. Kelas Bot

Kelas yang mengandung identitas *bot*(nama, email, id).

- Atribut

| Attribut       | Deskripsi                |
|----------------|--------------------------|
| name : str     | Nama pemilik <i>bot</i>  |
| id : str       | ID pemilik <i>bot</i>    |
| email : string | Email pemilik <i>bot</i> |

### 2. Kelas Position

Kelas yang merepresentasikan posisi objek di *board*

- Atribut

| Attribut | Deskripsi                     |
|----------|-------------------------------|
| y : int  | Ordinat objek di <i>board</i> |
| x : int  | Absis objek di <i>board</i>   |

### 3. Kelas Base

Kelas yang merepresentasikan posisi base. Kelas ini mewarisi atribut dari kelas **Position** dan tidak mengandung atribut atau *method* lain.

### 4. Kelas Properties

Kelas yang merepresentasikan properti-properti objek di *board*.

- Atribut

| Attribut                                | Deskripsi   |
|---|---|
| points: Optional[int] = None            | Poin objek di <i>board</i>                            |
| pair_id: Optional[str] = None           | <i>Identifier</i> unik sebuah pair                    |
| diamonds: Optional[int] = None          | Jumlah <i>diamond</i> yang dimiliki objek             |
| score: Optional[int] = None             | Jumlah skor milik objek                               |
| name: Optional[str] = None              | Nama objek  |
| inventory_size: Optional[int] = None    | Ukuran inventori objek                                |
| can_tackle: Optional[bool] = None       | Apakah objek dapat men- <i>tackle</i> objek(bot) lain |
| milliseconds_left: Optional[int] = None | Jumlah sisa waktu permainan                           |
| time_joined: Optional[str] = None       | Waktu objek bergabung dalam permainan                 |
| base: Optional[Base] = None             | Objek base yang dimiliki objek                        |

## 5. Kelas GameObject

Kelas yang merepresentasikan objek-objek yang terdapat di dalam permainan.

- Atribut

| Attribut                              | Deskripsi                    |
|---------------------------------------|------------------------------|
| id: int                               | ID objek di <i>board</i>     |
| position: Position                    | Posisi objek di <i>board</i> |
| type: str                             | Tipe objek di <i>board</i>   |
| properties: Optional[Properties]=None | Properti-properti objek      |

## 6. Kelas Config

Kelas yang merepresentasikan konfigurasi permainan.

- Atribut

| Attribut   | Deskripsi   |
|--|---|
| generation_ratio: Optional[float] = None         | Rasio generasi <i>diamond</i>                         |
| min_ratio_for_generation: Optional[float] = None | Minimum rasio generasi diamond                        |
| red_ratio: Optional[float] = None                | Rasio <i>diamond</i> merah dari semua <i>diamond</i>  |
| seconds: Optional[int] = None                    | Waktu permainan                                       |
| pairs: Optional[int] = None                      | Pair permainan  |
| inventory_size: Optional[int] = None             | Ukuran inventori <i>default</i> objek                 |
| can_tackle: Optional[bool] = None                | Apakah objek dapat men- <i>tackle</i> objek(bot) lain |

## 7. Kelas Feature

Kelas yang merepresentasikan fitur permainan.

- Atribut

| Attribut                        | Deskripsi                         |
|---------------------------------|-----------------------------------|
| name: str                       | Nama fitur                        |
| config: Optional[Config] = None | Konfigurasi fitur-fitur permainan |

## 8. Kelas Board

Kelas yang merepresentasikan *board* permainan

- Atribut

| Attribut                                    | Deskripsi  |
|---|--|
| id: int                                     | Rasio generasi <i>diamond</i>                        |
| width: int                                  | Minimum rasio generasi diamond                       |
| height: int                                 | Rasio <i>diamond</i> merah dari semua <i>diamond</i> |
| features: List[Feature]                     | Waktu permainan                                      |
| minimum_delay_between_moves: int            | Pair permainan                                       |
| game_objects:<br>Optional[List[GameObject]] | List dari objek-objek dalam permainan                |

- *Methods*

| Methods  | Deskripsi  |
|--|--|
| diamonds(self) -> List[GameObject]   | Mengembalikan list objek-objek bertipe <i>diamond</i>  |
| bots(self) -> List[GameObject]:  | Mengembalikan list objek-objek bot   |
| def get_bot(self, bot: Bot) -><br>Optional[GameObject]   | Mengembalikan <i>bot</i> milik pemain dalam bentuk GameObject, dari input <i>bot</i> milik pemain dalam bentuk Bot |
| is_valid_move(<br>self, current_position: Position,<br>delta_x: int, delta_y: int<br>) -> bool | Mengembalikan apakah <i>move</i> yang diinput merupakan <i>move</i> yang valid                                     |

## 9. Kelas BotHandler

Kelas yang menangani pergerakan dan informasi *bot*, serta kelas yang memasukkan *bot* ke dalam *board* dengan id yang sesuai.

- Atribut

| Atribut  | Deskripsi          |
|----------|--------------------|
| api: Api | Api dari permainan |

- Methods

| Methods   | Deskripsi  |
|---|--|
| _get_direction(dx: int, dy: int)  | Mengembalikan arah mata angin dari arah berbentuk integer ((1, 0), (0, 1), (-1, 0), (0, -1)) |
| get_my_info(self, token: str) -> Bot  | Mengembalikan informasi <i>bot</i> milik pemain dalam bentuk Bot                             |
| join(self, token: str, board_id: int) -> bool   | Mengembalikan respon dari percobaan join ke <i>board</i> .                                   |
| move(self, token: str, board_id: int, dx: int, dy: int) -> Optional[Board]                  | Mengembalikan keadaan <i>board</i> setelah <i>bot</i> melakukan <i>move</i> .                |
| register(<br>self, name: str, email: str,<br>password: str, team: str<br>) -> Optional[Bot] | Mengembalikan objek Bot yang berisi data yang didaftarkan.                                   |
| recover(self, email: str, password: str)<br>-> Optional[str]                                | Mengembalikan respon percobaan recover ke <i>api</i>   |

## 10. Kelas BoardHandler

Kelas yang menangani informasi *board-board* yang tersedia.

- Atribut

| Atribut  | Deskripsi          |
|----------|--------------------|
| api: Api | Api dari permainan |

- Methods

| Methods                                 | Deskripsi   |
|---|---|
| list_boards(self) -> List[Board]        | Mengembalikan list dari <i>board</i> yang tersedia              |
| get_board(self, board_id: int) -> Board | Mengembalikan <i>board</i> yang memiliki ID sesuai yang diinput |

## 11. Kelas Api

Kelas yang menangani interaksi dengan server permainan.

- Atribut

| Atribut  | Deskripsi     |
|----------|---------------|
| url: str | Url permainan |

- Methods

| Methods  | Deskripsi   |
|--|---|
| _get_url(self, endpoint: str) -> str   | Mengembalikan url dalam format yang tepat   |
| _req(self, endpoint: str, method: str, body: dict) -> Response                                       | Mengembalikan respon dari <i>request</i> ke server                                |
| bots_get(self, bot_token: str) -> Optional[Bot]  | Mengembalikan <i>bot</i> dalam bentuk Bot dari input token <i>bot</i> pada server |
| def bots_register(<br>self, name: str, email: str,<br>password: str, team: str<br>) -> Optional[Bot] | Mengembalikan objek Bot yang berisi data yang didaftarkan.                        |
| boards_list(self) -><br>Optional[List[Board]]  | Mengembalikan list <i>board</i> yang diterima dari server                         |

|  |   |
|--|---|
| bots_join(self, bot_token: str, board_id: int) -> bool   | Mengembalikan respon server dari percobaan join ke <i>board</i> |
| boards_get(self, board_id: str) -> Optional[Board]   | Mengembalikan board yang memiliki ID sesuai yang diinput        |
| bots_move(self, bot_token: str, direction: str) -> Optional[Board]                             | Mengembalikan keadaan board setelah bot melakukan move          |
| bots_recover(self, email: str, password: str) -> Optional[str]                                 | Mengembalikan respon percobaan recover ke server                |
| _return_response_and_status(<br>self, response: Response<br>) -> Tuple[Union[dict, List], int] | Mengembalikan hasil decoding respon server.                     |

## 12. Kelas BaseLogic

Kelas dasar dari logika *bot*.

- Methods

| Methods   | Deskripsi   |
|---|---|
| next_move(self, board_bot: GameObject, board: Board) -> Tuple[int, int] | Belum diimplementasikan. Akan diimplementasikan di kelas <i>child</i> . |

## 13. Kelas BotsMove

Kelas yang dibuat penulis untuk mencari move selanjutnya menggunakan algoritma *greedy*. Kelas ini mewarisi kelas BaseLogic.

- Atribut

| Atribut                 | Deskripsi                          |
|-------------------------|------------------------------------|
| directions: List[Tuple] | List gerakan-gerakan yang mungkin. |

|  |  |
|--|--|
| block_in: List[Tuple]                  | List zona-zona yang terdapat di <i>board</i> |
| goal_position: Optional[Position]=None | Posisi dari kotak tujuan <i>bot</i>          |

- Methods

| Methods   | Deskripsi   |
|---|---|
| __init__(self)  | Inisialisasi atribut  |
| boundary(self, n, smallest, largest)  | Memastikan n berada dalam boundary  |
| get_way(self, current_x, current_y, dest_x, dest_y, base, telestart:Position, teletarget:Position, checktele) | Mencari rute terbaik untuk mencapai tujuan (apakah menggunakan <i>teleport</i> )                      |
| chase(self, bot2:GameObject)  | Mengubah goal menjadi bot2  |
| get_tacklebot(self, board:Board, board_bot)   | Mengembalikan list <i>bot</i> lain yang dapat melakukan <i>tackle</i>                                 |
| get_redbut_telep(self, board, board_bot)  | Mengembalikan posisi <i>red button</i> dan <i>teleport</i>  |
| current_totalpointblock(self, current_position, diamond_objects)  | Mengembalikan total poin yang bisa didapatkan dari <i>diamond</i> di blok saat ini                    |
| bots_move(self, bot_token: str, direction: str) -> Optional[Board]  | Mengembalikan keadaan board setelah bot melakukan move  |
| totalpointblock(self,start_position,diamond_objects)  | Mengembalikan list blok yang ada dan total poin yang bisa didapatkan dari <i>diamond</i> di dalamnya. |
| closestDiamond(self,start_position,diamond_objects,Board:Board)   | Mengembalikan posisi <i>diamond</i> terdekat  |
| next_move(self, board_bot:  | Mengembalikan tujuan selanjutnya  |

|                           |                               |
|---------------------------|-------------------------------|
| GameObject, board: Board) | sesuai strategi <i>greedy</i> |
|---------------------------|-------------------------------|

#### 4.3. Pengujian Bot serta Analisis Performansi Bot Permainan Diamond

Pengujian dilakukan dengan melawan bot *custom* yang dibuat dengan fokus hanya mencari diamond terdekat bisa saja dilakukan dengan melawan *reference bot* yang diberikan dengan algoritma random, tetapi logika bot ini sangat buruk dan terkadang menghasilkan *invalid move*. Oleh karena itu kami membuat bot simple hanya dengan sepotong algoritma yang kita miliki untuk digunakan. Namun, dikarenakan permainan Etimo Diamond sangat random maka penilaian yang pasti juga tidak dapat diberikan. Kami memutuskan untuk melakukan 5 kali percobaan normal(`sleep(1)`) dan 2 percobaan *custom* (`sleep(0.1)`) untuk menganalisis kinerja bot yang telah dibuat. Bot dengan algoritma greedy dinamakan ‘main’ dan bot *custom* dinamakan ‘referensi’. Berikut hasil akhir match match tersebut:

- Match 1

| Board 10 players |          |       |      |
|------------------|----------|-------|------|
| Name             | Diamonds | Score | Time |
| main             | ◆◆◆◆◆    | 10    | 0s   |
| referensi        | ◆◆◆◆     | 9     | 1s   |

Perbedaan point sangat tipis hanya 1 diamond.

- Match 2

| Final Score |       |
|-------------|-------|
| Name        | Score |
| main        | 12    |
| referensi   | 10    |

Perbedaan point masih sangat tipis hanya 2 diamonds.

- Match 3

| Final Score |       |
|-------------|-------|
| Name        | Score |
| main        | 6     |
| referensi   | 4     |

Perbedaan point hanya 2 diamonds.

- Match 4

| Final Score |       |
|-------------|-------|
| Name        | Score |
| main        | 4     |
| referensi   | 4     |

Tidak ada perbedaan point

- Match 5

| Final Score |       |
|-------------|-------|
| Name        | Score |
| refrensi    | 13    |
| mainbot     | 4     |

bot utama “mainbot” kalah telak dengan perbedaan 9 point.

- Match 6 (custom)

| Final Score |       |
|-------------|-------|
| Name        | Score |
| mainbot     | 92    |
| referensi   | 88    |

bot utama menang dengan perbedaan 4 point.

- Match 7 (custom)

| Final Score |       |
|-------------|-------|
| Name        | Score |
| referensi   | 93    |
| main        | 78    |

bot utama kalah dengan perbedaan 15 point

Setelah 5 pengujian normal dan 2 percobaan *custom*, hasil yang didapat adalah bot kami berhasil memenangkan permainan 5 dari 7 kali percobaan. Dapat dilihat juga pada score akhir, jarak antara kedua bot tidak pernah jauh sehingga bisa disimpulkan bahwa pertandingan antara kedua *bot* sangat sengit. Dari sini didapat bahwa algoritma *Greedy* yang kami buat cukup optimal dan dapat mengalahkan bot lain, dilihat dengan persentase kemenangan sebesar 70%. Meskipun begitu masih terdapat beberapa case dimana bot kami mengalami error, baik dari bug game ataupun kesalahan algoritma. Kami sudah berusaha untuk menghandle agar dapat menghindari fitur teleport, tetapi ada case dimana terjadi looping yang menyebabkan bot kami menjadi bergerak tidak menentu karena berusaha menghindari teleport tersebut. Terutama ketika teleport berada diarah jalan pulang menuju base. Pada normalnya bot kami cukup mandiri karena dapat selalu menentukan jalan yang menguntungkan dan apabila terdapat red button serta total diamond pada block bot berada 0, maka ia akan memilih jalan tersebut. Pada match 1-4. Bot kami dapat selalu menentukan jalan yang terbaik, tetapi karena terdapat jeda selama 1 detik pada program maka hasilnya tidak bisa terlalu banyak. Pada match 5 dan 7 kami menemukan bug teleport tersebut dimana bot kami berada pada posisi yang tidak menguntungkan dan stuck di teleport sehingga bot kami menghabiskan banyak waktu dan tidak dapat menjalankan fungsinya. Pada match 6 bot kami berjalan sangat baik dan implementasi greedy yang diberikan sangat bermanfaat.

## Bab 5: Kesimpulan dan Saran

### 5.1. Kesimpulan

Kelompok kami berhasil mengimplementasikan algoritma *greedy* untuk membuat bot permainan Diamond yang bisa mencapai tujuan objektif yaitu mendapat skor tertinggi. Dari hasil yang didapatkan, dapat dilihat bahwa penggunaan strategi *greedy* cukup optimal dalam kasus ini, dikarenakan pada permainan Diamond pemilihan paling tamak pada tiap langkahnya kemungkinan besar merupakan kemungkinan terbaik untuk permainan secara umum.

Namun, tentu strategi *greedy* ini juga perlu didampingi dengan teknik heuristik, agar strategi *greedy* yang digunakan lebih optimal dan cocok untuk berbagai kondisi. Secara umum, *bot* yang kami buat berhasil membuktikan bahwa strategi *greedy* cukup baik digunakan dalam pembuatan bot, didukung dengan keberhasilan pengujian melawan bot – bot lain yang ada.

### 5.2. Saran

Terkait dengan topik ini, berikut beberapa saran yang bisa kita ajukan untuk selanjutnya:

- Alangkah baiknya dilakukan pembagian tugas terlebih dahulu agar *workload* masing – masing anggota jelas dan penggerjaan lebih terstruktur
- Penulis menyarankan untuk berikutnya, pembuatan laporan dapat dilakukan lebih cepat lagi dan tidak terlalu mepet dengan tanggal pengumpulan
- Terakhir, pengujian yang dilakukan melawan bot lain bisa dibuat lebih adil lagi. Salah satunya dengan membuat map statis. Dilihat dari banyaknya faktor *luck* yang lumayan berpengaruh pada permainan.

## **Daftar Pustaka**

[informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm)

## TERLAMPIR

Link Github Kode: [EkaaPrawiraa/Tubes-1-STIMA \(github.com\)](https://github.com/EkaaPrawiraa/Tubes-1-STIMA)

Link Video: <https://youtu.be/3RHsDnuyitU?si=BKpyygqrU6OwevLz>