PEMANFAATAN ALGORITMA *GREEDY* DALAM PEMBUATAN BOT PERMAINAN DIAMONDS

Oleh:

Kelompok Cijawjaw

1. 13522128 - Mohammad Andhika Fadillah
2. 13522145 - Farrel Natha Saskoro
3. 13522156 - Jason Fernando

IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA

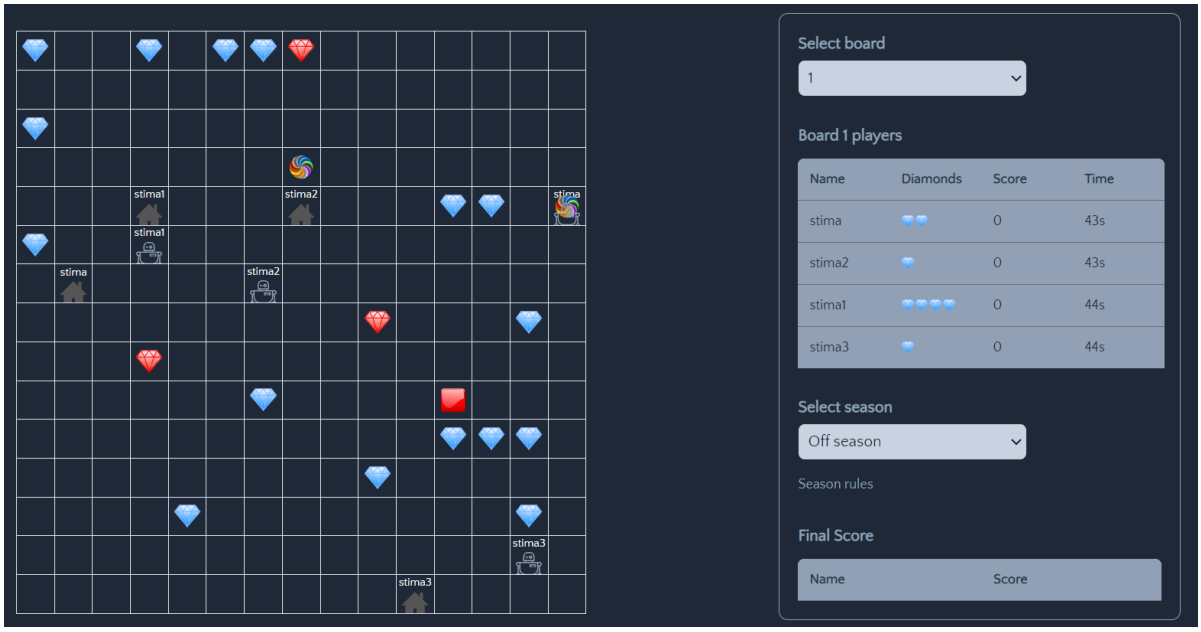
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

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



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

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*.

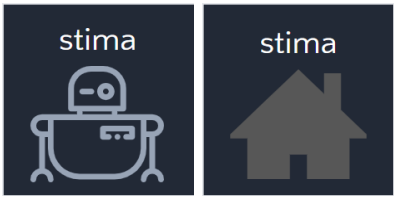
2. **Red Button/Diamond Button**

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**

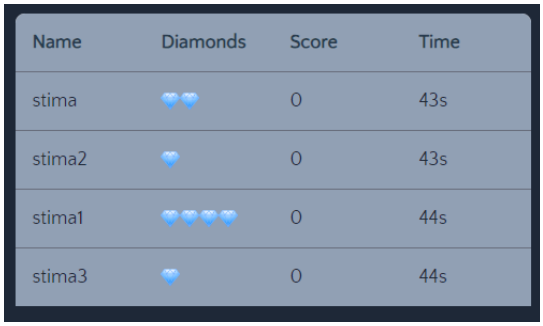
Terdapat 2 *teleporter* yang saling terhubung satu sama lain. Jika bot melewati sebuah *teleporter* maka bot akan berpindah menuju posisi *teleporter* yang lain.

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

BAB 2
TEORI DASAR

2.1 Algoritma Greedy

Algoritma *greedy* adalah algoritma yang paling populer dan sederhana untuk memecahkan persoalan optimasi. Persoalan optimasi hanya memiliki 2 macam persoalan, yaitu maksimasi dan minimasi. Algoritma greedy memecahkan persoalan optimasi secara langkah per langkah sedemikian sehingga, pada setiap langkah, diambil pilihan yang terbaik pada saat itu tanpa memperhatikan konsekuensi ke depan dan berharap dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimal.

Elemen - elemen pada algoritma greedy :

1. Himpunan Kandidat (C) : berisi kandidat yang akan dipilih pada setiap langkah algoritma.
2. Himpunan Solusi : berisi kandidat yang sudah dipilih.
3. Fungsi Solusi : menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi Seleksi (*selection function*) : memilih kandidat berdasarkan strategi greedy tertentu dan strategi bersifat heuristik.
5. Fungsi Kelayakan (*feasible*) : Memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak).
6. Fungsi Obyektif : mencari solusi yang paling efektif.

Contoh - contoh persoalan yang diselesaikan dengan algoritma greedy antara lain adalah persoalaan penukaran uang, Fractional Knapsack, minimasi waktu di dalam sistem (penjadwalan), dan masih banyak lagi.

2.2 Game Engine

Game Engine adalah *software* yang dirancang untuk menciptakan dan mengembangkan sebuah game. *Game Engine* pada dasarnya adalah sebuah *library* yang dapat digunakan untuk membuat permainan. *Game Engine* dapat digunakan untuk membuat lebih dari satu permainan, dan pengembang permainan dapat

mengoptimalisasi proses pengembangan dengan cara menggunakan atau mengadaptasi *game engine* yang telah ada sebelumnya. Contoh - contoh fungsi yang dapat membantu game *developer* adalah mesin render 2D atau 3D, suara, *script*, animasi, kecerdasan buatan, jaringan, *streaming*, dan masih banyak lagi. Dengan adanya *game engine*, para *developer* tidak harus membuat game dari awal.

2.3 Bagaimana Bot melakukan aksinya

Bot dalam permainan Diamonds berperan sebagai entitas yang dilengkapi dengan beragam aksi dasar, semua yang dirancang secara strategis untuk mencapai tujuan utamanya: mengumpulkan diamond sebanyak mungkin. Sejalan dengan kompleksitas permainan, bot harus menguasai serangkaian interaksi yang melibatkan pergerakan, pengambilan diamond, serta keterlibatan dengan objek-objek permainan seperti diamond itu sendiri, red button, teleporter, dan base, termasuk manajemen inventory. Efisiensi navigasi menjadi kunci, di mana bot harus mampu melintasi peta permainan dengan cerdas, mengambil diamond dengan presisi, menghindari pertemuan dengan bot lawan, dan bijaksana menggunakan fitur tambahan seperti teleporter.

Dalam memahami dinamika permainan Diamonds, penting bagi bot untuk mempertimbangkan aspek-aspek seperti penentuan rute terbaik, penanganan red button untuk meregenerasi diamond, penggunaan teleporter dengan strategi yang efisien, dan manajemen inventory yang optimal. Pemilihan langkah-langkah ini harus dilakukan dengan mempertimbangkan nilai diamond, waktu yang tersisa, serta potensi interaksi dengan bot lawan. Dengan menguasai semua elemen ini, bot dapat mengoptimalkan peluangnya dalam meraih skor tertinggi pada setiap pertandingan Diamonds yang dihadapinya.

2.4 bagaimana mengimplementasikan algoritma greedy ke dalam bot

Implementasi algoritma greedy pada bot Diamonds melibatkan serangkaian keputusan strategis yang diambil pada setiap langkah permainan. Keputusan ini difokuskan pada pemaksimalan jumlah diamond yang dapat dikumpulkan, dengan mempertimbangkan beberapa aspek penting.

Langkah pertama dalam algoritma greedy adalah pemilihan rute terbaik untuk mencapai diamond. Bot harus secara cerdas mengeksplorasi peta permainan, mengidentifikasi diamond biru dan merah, dan memilih rute yang memberikan akses tercepat dan teraman ke diamond tersebut. Pengambilan keputusan yang efisien dalam hal ini akan memberikan keuntungan signifikan terhadap pemain lawan.

Penanganan red button menjadi langkah krusial dalam implementasi algoritma greedy. Bot perlu memutuskan kapan dan bagaimana melewati red button untuk memicu regenerasi

diamond. Keputusan ini harus dilakukan dengan hati-hati, mempertimbangkan manfaat regenerasi diamond tanpa kehilangan waktu yang berlebihan atau mengorbankan posisi strategis.

Penggunaan teleporter juga menjadi bagian dari strategi greedy. Bot harus mampu mengidentifikasi teleporter yang dapat meningkatkan efisiensi pergerakan, memungkinkan akses cepat ke area-area yang kaya diamond. Keputusan untuk menggunakan teleporter harus didasarkan pada analisis posisi saat ini dan potensi keuntungan yang dapat diperoleh.

Implementasi algoritma greedy membekali bot dengan kecerdasan untuk membuat keputusan yang tepat pada setiap langkah permainan. Dengan memprioritaskan rute terbaik, penanganan red button, penggunaan teleporter yang efisien, dan manajemen inventory yang cerdas, bot dapat menjadi pesaing tangguh dalam pertandingan Diamonds dengan tujuan akhir memenangkan permainan.

2.5 bagaimana menjalankan bot

Untuk menjalankan satu bot dengan logic yang terdapat pada file diperlukan yang telah dibuat gunakan perintah seperti di bawah ini.

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

Untuk menjalankan beberapa bot sekaligus, misalnya menjalankan 4 bot dengan logic yang sama. Perlu di buat terlebih dahulu file *run-bots.bat* pada Windows. File tersebut akan berisi perintah - perintah yaitu sebagai berikut.

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

BAB 3
APLIKASI STRATEGI *GREEDY*

3.1 Mapping persoalan Diamonds menjadi elemen-elemen algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan Diamonds selain bot player.
Himpunan Solusi	Diamond yang menjadi target pencarian
Fungsi Solusi	Mengarahkan bot untuk mencapai target (diamond)
Fungsi Seleksi	Melakukan seleksi dari semua objek yang ada di permainan yang sesuai dengan strategi *greedy* yang dipakai.
Fungsi Kelayakan	Melakukan validasi jika diamond sudah mecapai kapasitas maksimum dari bot yang dipakai.
Fungsi Obyektif	Mempertahankan diri agar tidak terkena tackle dan mengumpulkan diamond sebanyak - banyaknya

3.2 Eksplorasi alternatif solusi greedy yang mungkin dipilih dalam persoalan Diamonds

1. Alternatif Solusi 1
   a. Penjelasan Umum Strategi

      Salah satu strategi yang dapat digunakan adalah algoritma *greedy by distance*. Greedy by distance adalah pendekatan algoritma greedy yang memprioritaskan jarak antara bot dan diamonds sebagai pendekatannya.

Bot akan berusaha mencari diamonds terdekat dan memprioritaskan diamond yang terdekat itu untuk diambil.

b. Mapping Elemen Algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan diamonds selain bot player.
Fungsi Kelayakan	Menyaring objek yang bukan merupakan diri sendiri
Fungsi Seleksi	Mengurutkan bot dari yang terdekat sampai yang terjauh
Fungsi Solusi	Mengambil diamonds terdekat untuk dijadikan sasaran tujuan
Himpunan Solusi	diamond yang menjadi target
Fungsi Obyektif	Menghindari agar tidak di tackle oleh bot lawan

2. Alternatif Solusi 2

a. Penjelasan Umum Strategi

Strategi yang kami pilih yaitu algoritma *greedy by distance and to attack*. *greedy by distance and to attack* adalah pendekatan algoritma greedy yang memprioritaskan jarak antara pemain sebagai pendekatannya akan tetapi, algoritma ini memiliki tambahan fitur attack dimana bot akan mendeteksi bot dengan jumlah diamond lebih dari 0, lalu bot juga akan memprioritaskan pengambilan diamond jika jarak antar bot dan diamond adalah 1.

b. Mapping Elemen Algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan diamonds selain bot player.
Fungsi Kelayakan	Menyaring objek yang bukan merupakan diri sendiri
Fungsi Seleksi	Mengurutkan bot dari yang terdekat sampai yang terjauh
Fungsi Solusi	Mengambil diamonds terdekat untuk dijadikan sasaran tujuan dan men-*tackle* bot lawan
Himpunan Solusi	diamond yang menjadi target
Fungsi Obyektif	Menghindari agar tidak di tackle oleh bot lawan

3. Alternatif Solusi 3

    a. Penjelasan Umum Strategi

        Strategi yang kami pilih yaitu algoritma *greedy by distance and to attack*. *greedy by distance and to attack* adalah pendekatan algoritma greedy yang memprioritaskan jarak antara pemain sebagai pendekatannya akan tetapi, algoritma ini memiliki tambahan fitur avoid teleport dimana bot akan menghindari teleport jika bot ingin kembali ke base.

    b. Mapping Elemen Algoritma Greedy

Himpunan Kandidat	Semua objek yang ada di dalam permainan diamonds selain bot player.
Fungsi Kelayakan	Menyaring objek yang bukan merupakan diri sendiri
Fungsi Seleksi	Mengurutkan bot dari yang terdekat sampai yang terjauh
Fungsi Solusi	Mengambil diamonds terdekat untuk dijadikan sasaran tujuan dan menghindari teleport pada saat bot ingin kembali ke base.
Himpunan Solusi	diamond yang menjadi target dan men-*tackle* bot lawan
Fungsi Obyektif	Menghindari agar tidak di tackle oleh bot lawan

3.3 Analisis efisiensi dan efektivitas dari kumpulan alternatif solusi

Dari tiga opsi solusi yang telah kami kembangkan, dapat disimpulkan bahwa solusi yang paling optimal terletak pada strategi di mana bot memprioritaskan pencarian diamond terdekat dan mengabaikan tindakan bot lawan. Melalui lima percobaan yang dilakukan, terungkap bahwa strategi ini memberikan hasil yang sangat baik, dengan bot berhasil meraih kemenangan sebanyak empat kali dari lima percobaan tersebut.

Pendekatan di mana bot fokus pada pencarian diamond terdekat dan mengesampingkan tindakan bot lawan tampaknya memberikan keunggulan yang signifikan. Keberhasilan ini menunjukkan bahwa strategi ini mungkin lebih handal dan efektif dalam konteks permainan Diamonds. Meskipun demikian, perlu dicatat bahwa evaluasi terus-menerus dan pengujian lebih lanjut mungkin diperlukan untuk memastikan konsistensi dan keandalan strategi ini dalam berbagai skenario permainan.

3.4 Pemilihan Strategi Greedy

Dari berbagai alternatif strategi greedy yang telah diuraikan sebelumnya, dapat dirancang sebuah algoritma greedy yang menggabungkan elemen-elemen terbaik dari setiap strategi. Pendekatan ini dirancang untuk mengatasi kekurangan atau kelemahan yang mungkin muncul dalam satu strategi tertentu dengan memanfaatkan keunggulan lainnya. Algoritma ini bertujuan untuk menciptakan sebuah bot yang tangguh dan adaptif dalam berbagai situasi permainan Diamonds. Pendekatan ini memberikan fleksibilitas yang diperlukan untuk membuat keputusan cerdas yang dapat memberikan keunggulan strategis dalam pertandingan, mengoptimalkan pengumpulan diamond, dan meningkatkan peluang kemenangan bot.

Algoritma *greedy by distance* yang digunakan untuk mengambil dan mengamankan diamond yang jaraknya terdekat dari bot

Algoritma *greedy by distance and to attack* yang digunakan untuk mengambil diamond yang terdekat dari bot tetapi memiliki fitur tambahan yaitu dapat mengejar dan men-*tackle* lawan.

Algoritma greedy by distance and to avoid teleport yang digunakan untuk mengambil dan mengamankan diamond yang jaraknya terdekat dari bot. Fitur avoid teleport akan terpakai apabila pada saat bot ingin kembali ke base dan bertemu dengan teleport, bot tersebut akan menghindari teleport tersebut untuk kembali berjalan ke base..

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi algoritma greedy pada program bot yang digunakan

Greedy.py

```
class GreedyLogic extends BaseLogic:
    directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    goal_position = null
    current_direction = 0

    method GreedyLogic():
        initialize directions, goal_position as null, current_direction

    method next_move(board_bot: GameObject, board: Board):
        props = board_bot.properties
        current_position = board_bot.position

        button = find_object_by_type(board.game_objects,
"DiamondButtonGameObject")
        button_distance = abs(button.position.x - current_position.x) +
abs(button.position.y - current_position.y)

        teleports = find_objects_by_type(board.game_objects, "TeleportGameObject")
        teleport1 = teleports[0]
        teleport2 = teleports[1]

        if props.diamonds == 5:
            base = board_bot.properties.base
            goal_position = base
        else:
            goal_position = null

        current_position = board_bot.position
        if goal_position:
            delta_x, delta_y = get_direction(current_position.x, current_position.y,
goal_position.x, goal_position.y)
        else:
            diamonds_queue = initialize_priority_queue(board.diamonds, current_position)

            if current_position == props.base:
                min_distance, nearest_diamond_index = heapq.heappop(diamonds_queue)
                goal_position = board.diamonds[nearest_diamond_index].position
            else:
                if props.diamonds == 4:
                    min_distance, nearest_diamond_index =
```

```
heapq.heappop(diamonds_queue)
                if (abs(props.base.x - current_position.x) + abs(props.base.y -
current_position.y)) < min_distance and props.diamonds != 0:
                    goal_position = props.base
                else:
                    if board.diamonds[nearest_diamond_index].properties.points == 1:
                        goal_position = board.diamonds[nearest_diamond_index].position
                    else:
                        while board.diamonds[nearest_diamond_index].properties.points != 1
and diamonds_queue:
                            min_distance, nearest_diamond_index =
heapq.heappop(diamonds_queue)
                        goal_position = board.diamonds[nearest_diamond_index].position
            else:
                min_distance, nearest_diamond_index =
heapq.heappop(diamonds_queue)
                if (abs(props.base.x - current_position.x) + abs(props.base.y -
current_position.y)) < min_distance and props.diamonds != 0:
                    goal_position = props.base
                elif button_distance < min_distance:
                    goal_position = button.position
                else:
                    goal_position = board.diamonds[nearest_diamond_index].position

            if (props.milliseconds_left / 1000) <= (abs(props.base.x - current_position.x)
+ abs(props.base.y - current_position.y) + 2) and props.diamonds != 0:
                goal_position = props.base

          if (props.milliseconds_left / 1000) < 2 and current_position == props.base:
              delta_x, delta_y = 0, 1

          if goal_position == props.base:
              teleport_in_path = is_teleport_in_path(current_position, props.base,
teleport1, teleport2)
              if teleport_in_path:
                  goal_position = get_opposite_position(current_position, teleport_in_path)

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

       return delta_x, delta_y

    method is_teleport_in_path(start_position: Position, end_position: Position,
teleport1: GameObject, teleport2: GameObject) -> Optional[GameObject]:
       path_direction = get_direction(start_position.x, start_position.y, end_position.x,
end_position.y)

       if is_between(start_position, teleport1.position, end_position, path_direction):
          return teleport1
       elif is_between(start_position, teleport2.position, end_position, path_direction):
```

```
            return teleport2
        else:
            return null

    method is_between(start_position: Position, middle_position: Position, end_position:
Position, path_direction: Tuple[int, int]) -> bool:
        return (
            start_position.x <= middle_position.x <= end_position.x or start_position.x >=
middle_position.x >= end_position.x
        ) and (
            start_position.y <= middle_position.y <= end_position.y or start_position.y >=
middle_position.y >= end_position.y
        )

    method get_opposite_position(current_position: Position, teleport_in_path:
GameObject) -> Position:
        teleport_position = teleport_in_path.position
        delta_x, delta_y = get_direction(current_position.x, current_position.y,
teleport_position.x, teleport_position.y)
        new_goal_position = Position(teleport_position.x + delta_x, teleport_position.y +
delta_y)
        return new_goal_position

    method find_object_by_type(game_objects: List[GameObject], object_type: str) ->
GameObject:
        objects = [obj for obj in game_objects if obj.type == object_type]
        return objects[0]

    method find_objects_by_type(game_objects: List[GameObject], object_type: str) ->
List[GameObject]:
        objects = [obj for obj in game_objects if obj.type == object_type]
        return
```

Attack.py

```
class AttackLogic extends BaseLogic:
    directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    goal_position = null
    current_direction = 0

    method AttackLogic():
        initialize directions, goal_position as null, current_direction

    method next_move(board_bot: GameObject, board: Board):
        props = board_bot.properties
        current_position = board_bot.position

        nearby_enemy_bots = []
        for enemy_bot in board.bots:
            if enemy_bot != board_bot and is_nearby(current_position,
```

```
enemy_bot.position):
            nearby_enemy_bots.append(enemy_bot)

        if nearby_enemy_bots:
            enemy_bots_with_diamonds = []
            for enemy_bot in nearby_enemy_bots:
                if enemy_bot.properties.diamonds > 0:
                    enemy_bots_with_diamonds.append(enemy_bot)

            if enemy_bots_with_diamonds:
                nearest_enemy_bot = min(enemy_bots_with_diamonds, key=lambda bot:
calculate_distance(current_position, bot.position))
                goal_position = nearest_enemy_bot.position
            else:
                handle_no_enemy_with_diamonds(current_position, props, board,
nearby_enemy_bots)
        elif props.diamonds == 5:
            goal_position = props.base
        else:
            goal_position = find_nearest_diamond(current_position, board.diamonds)

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

    method handle_no_enemy_with_diamonds(current_position: Position, props, board,
nearby_enemy_bots):
        enemy_bots_with_5_diamonds = []
        for enemy_bot in nearby_enemy_bots:
            if enemy_bot.properties.diamonds == 5:
                enemy_bots_with_5_diamonds.append(enemy_bot)

        if enemy_bots_with_5_diamonds:
            goal_position = enemy_bots_with_5_diamonds[0].properties.base
        else:
            goal_position = find_nearest_diamond(current_position, board.diamonds)

    method is_nearby(position1: Position, position2: Position, distance_threshold: int =
1) -> bool:
        distance = abs(position1.x - position2.x) + abs(position1.y - position2.y)
        return distance <= distance_threshold

    method calculate_distance(position1: Position, position2: Position) -> int:
        return abs(position1.x - position2.x) + abs(position1.y - position2.y)

    method find_nearest_diamond(current_position: Position, diamonds:
List[GameObject]) -> Position:
        min_distance = infinity
        nearest_diamond_position = null
        for diamond in diamonds:
```

```
        distance = calculate_distance(current_position, diamond.position)
        if distance < min_distance:
            min_distance = distance
            nearest_diamond_position = diamond.position

    return nearest_diamond_position
```

Ultimate.py

```
class GreedyLogicUltimate extends BaseLogic:
    directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    goal_position = null
    current_direction = 0

    method GreedyLogicUltimate():
        initialize directions, goal_position as null, current_direction

    method next_move(board_bot: GameObject, board: Board):
        props = board_bot.properties
        current_position = board_bot.position

        button = find_object_by_type(board.game_objects,
"DiamondButtonGameObject")
        button_distance = abs(button.position.x - current_position.x) +
abs(button.position.y - current_position.y)

        teleports = find_objects_by_type(board.game_objects, "TeleportGameObject")
        teleport1 = teleports[0]
        teleport2 = teleports[1]

        nearby_enemy_bots = find_nearby_enemy_bots(board.bots, board_bot,
current_position)

        if props.diamonds == 5:
            goal_position = props.base
        else:
            goal_position = null

        if goal_position:
            delta_x, delta_y = get_direction(current_position.x, current_position.y,
goal_position.x, goal_position.y)
        else:
            diamonds_queue = initialize_priority_queue(board.diamonds, current_position)

            if current_position == props.base:
                min_distance, nearest_diamond_index = heapq.heappop(diamonds_queue)
                goal_position = board.diamonds[nearest_diamond_index].position
            else:
                if props.diamonds == 4:
                    if nearby_enemy_bots:
```

```
                    nearest_enemy_bot = find_nearest_enemy_bot(nearby_enemy_bots,
current_position)
                    goal_position = nearest_enemy_bot.position
                else:
                    min_distance, nearest_diamond_index =
heapq.heappop(diamonds_queue)
                    if (abs(props.base.x - current_position.x) + abs(props.base.y -
current_position.y)) < min_distance and props.diamonds != 0:
                        goal_position = props.base
                    else:
                        if board.diamonds[nearest_diamond_index].properties.points == 1:
                            goal_position = board.diamonds[nearest_diamond_index].position
                        else:
                            while board.diamonds[nearest_diamond_index].properties.points
!= 1 and diamonds_queue:
                                min_distance, nearest_diamond_index =
heapq.heappop(diamonds_queue)
                            goal_position = board.diamonds[nearest_diamond_index].position
            else:
                min_distance, nearest_diamond_index =
heapq.heappop(diamonds_queue)
                if nearby_enemy_bots:
                    nearest_enemy_bot = find_nearest_enemy_bot(nearby_enemy_bots,
current_position)
                    goal_position = nearest_enemy_bot.position
                else:
                    if (abs(props.base.x - current_position.x) + abs(props.base.y -
current_position.y)) < min_distance and props.diamonds != 0:
                        goal_position = props.base
                    elif button_distance < min_distance:
                        goal_position = button.position
                    else:
                        goal_position = board.diamonds[nearest_diamond_index].position

            if (props.milliseconds_left / 1000) <= (abs(props.base.x - current_position.x)
+ abs(props.base.y - current_position.y) + 2):
                if nearby_enemy_bots:
                    nearest_enemy_bot = find_nearest_enemy_bot(nearby_enemy_bots,
current_position)
                    goal_position = nearest_enemy_bot.position
                else:
                    goal_position = props.base

            if (props.milliseconds_left / 1000) < 10 and current_position == props.base:
                goal_position = null

            if goal_position == props.base:
                teleport_in_path = is_teleport_in_path(current_position, props.base,
teleport1, teleport2)
                if teleport_in_path:
```

```
                goal_position = get_opposite_position(current_position, teleport_in_path)

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

        return delta_x, delta_y

    method find_nearby_enemy_bots(bots: List[GameObject], board_bot: GameObject,
current_position: Position) -> List[GameObject]:
        nearby_enemy_bots = []
        for enemy_bot in bots:
            if enemy_bot != board_bot and is_nearby(current_position,
enemy_bot.position):
                nearby_enemy_bots.append(enemy_bot)
        return nearby_enemy_bots

    method find_object_by_type(game_objects: List[GameObject], object_type: str) ->
GameObject:
        objects = [obj for obj in game_objects if obj.type == object_type]
        return objects[0]

    method find_objects_by_type(game_objects: List[GameObject], object_type: str) ->
List[GameObject]:
        objects = [obj for obj in game_objects if obj.type == object_type]
        return objects

    method initialize_priority_queue(diamonds: List[GameObject], current_position:
Position) -> List[Tuple[int, int]]:
        diamonds_queue = []
        for index, diamond in enumerate(diamonds):
            distance = abs(diamond.position.x - current_position.x) +
abs(diamond.position.y - current_position.y)
            heapq.heappush(diamonds_queue, (distance, index))
        return diamonds_queue

    method find_nearest_enemy_bot(enemy_bots: List[GameObject], current_position:
Position) -> GameObject:
        nearest_enemy_bot = min(enemy_bots, key=lambda bot:
calculate_distance(current_position, bot.position))
        return nearest_enemy_bot
```

4.2 Penjelasan struktur data yang digunakan dalam program bot Diamonds.

Berikut adalah rincian struktur data dan variabel yang dipakai dalam program.

1. Position

Kelas ini menyimpan informasi tentang posisi dalam permainan diamonds, yang terdiri dari koordinat x dan y.

```
class Position:
    y: integer
    x: integer
```

2. Game Object

Kelas ini menyimpan Properties yang digunakan untuk menyimpan properti tambahan dari objek yang berupa poins, pair_id, diamonds, score, name, ukuran inventory, sisa waktu dalam milidetik, waktu bergabung, dan posisi *base*.

```
class Properties:
    points: integer = None
    pair_id: string = None
    diamonds: integer = None
    score: integer = None
    name: string = None
    inventory_size: integer = None
    can_tackle: boolean = None
    milliseconds_left: integer = None
    time_joined: string = None
    base: Base = None

class GameObject:
    id: integer
    position: Position
    type: string
    properties: Properties = None
```

3. Variabel

Berikut adalah penjelasan dari masing-masing variabel dan fungsinya dalam program.

a. *static_goals*: List yang menyimpan posisi tujuan-tujuan yang ingin dicapai oleh bot.

b. static_goal_teleport: Objek yang menyimpan informasi tentang *teleporter* yang menjadi tujuan bot.
c. static_temp_goals: Sebuah objek yang menampung data mengenai lokasi akhir sementara. Fungsinya adalah untuk menyimpan sementara posisi tujuan ketika bot perlu menghindari rintangan di sepanjang jalurnya.
d. *Static_direct_to_base_via_teleporter*: Sebuah variabel boolean yang menunjukkan apakah bot perlu segera menuju ke basis menggunakan teleporter. Jika nilainya True, bot akan berusaha langsung menuju basis melalui teleporter.
e. *goal_position*: Variabel yang menyimpan posisi yang ingin dicapai oleh bot dalam langkah selanjutnya.
f. *distance*: Variabel yang menyimpan jarak antara bot dan *goal.*
g. *props*: Objek yang menyimpan properti-properti yang terkait dengan bot.
h. *board*: List yang berisi semua objek dalam papan permainan.
i. *board_bot*: Objek yang mewakili bot dalam *game*.
j. *diamonds*: List yang berisi semua objek *diamonds* dalam permainan diamonds.
k. *bots*: List yang berisi semua objek bot dalam permainan, termasuk bot-bot musuh juga.
l. *teleporter*: List yang berisi *teleporter* dalam permainan.
m. *redButton*: List yang berisi *red button* dalam permainan.
n. *enemy*: List yang berisi semua bot musuh dalam permainan.
o. *enemyDiamond*: List yang berisi informasi tentang jumlah *diamonds* yang dikumpulkan oleh bot musuh.

```
static_goals : array of Position = []
static_goal_teleport : GameObject = None
static_temp_goals : Position = None
static_direct_to_base_via_teleporter : boolean = False
goal_position: Position = None
distance: int = 0
props: Properties
board: array of GameObject
board_bot: GameObject
Diamonds: array of GameObject
bots: array of GameObject
teleporter: array of GameObject
redButton: array of GameObject
```

```
enemy: array of GameObject
enemyDiamond: array of GameObject
```

4.3 Analisis dari desain solusi algoritma greedy yang diimplementasikan

Desain solusi menggunakan algoritma greedy, meskipun menunjukkan kemampuan yang baik dalam mengumpulkan diamonds dan menghindari obstacle pada kasus tertentu, belum selalu optimal ketika dihadapkan pada situasi yang lebih kompleks. Secara keseluruhan, kinerja bot ini telah memberikan hasil yang memuaskan dalam mode permainan tanpa kehadiran musuh, berhasil mengoptimalkan perolehan diamonds dan mengelak dari rintangan.

Namun, ketika algoritma bot diuji dalam kondisi nyata dengan kehadiran bot musuh, terlihat variasi kinerja yang lebih signifikan dan potensi terjadinya kesalahan. Hal ini menunjukkan bahwa solusi yang telah dikembangkan belum sepenuhnya dapat menangani kompleksitas yang muncul ketika berhadapan dengan pemain lain.

Untuk meningkatkan performa dan kestabilan bot, diperlukan pengembangan lebih lanjut, terutama dalam menanggapi skenario-skenario yang tidak terduga, seperti situasi di mana bot terkena tackle oleh lawan. Mengidentifikasi dan mengatasi kelemahan-kelemahan ini akan menjadi kunci untuk meningkatkan daya adaptasi dan responsivitas bot dalam menghadapi tantangan yang lebih bervariasi dan dinamis dalam permainan Diamonds.

BAB 5
PENUTUP

5.1 Kesimpulan

Kesimpulan dari penerapan algoritma greedy dalam menyelesaikan permainan Diamonds dapat dijelaskan sebagai berikut: Algoritma greedy memiliki efisien yang tinggi. Pendekatan ini memungkinkan bot untuk membuat keputusan berdasarkan langkah yang paling menguntungkan pada setiap fase permainan, termasuk penggunaan teleporter secara optimal dan menghindari rintangan.

Performa algoritma greedy dapat dipengaruhi oleh kehadiran lawan dalam permainan. Meskipun demikian, masih terdapat ruang untuk pengembangan lebih lanjut. Fokus pengembangan sebaiknya diberikan pada penanganan kasus-kasus khusus, seperti penanganan kesalahan saat bot terkena serangan. Melalui pengembangan ini, diharapkan performa dan stabilitas algoritma greedy dapat ditingkatkan.

5.2 Saran

Beberapa langkah dapat diambil untuk meningkatkan kinerja dan stabilitas algoritma greedy dalam menyelesaikan permainan Diamonds. Pertama, pengembangan harus lebih fokus pada penanganan kasus-kasus khusus yang berpotensi menyebabkan kesalahan pada bot. Kedua, algoritma bot seharusnya lebih mempertimbangkan kehadiran musuh dan memprediksi kemungkinan kejadian saat berhadapan dengan mereka. Ketiga, pengujian algoritma bot perlu diperluas ke berbagai skenario permainan untuk mengidentifikasi potensi kelemahan dan kesalahan yang memerlukan perbaikan. Dengan menerapkan langkah-langkah tersebut, diharapkan algoritma greedy dapat mengalami peningkatan kinerja yang lebih baik dalam menyelesaikan permainan Diamonds.

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/fnathas/Tubes1_Cijawjaw

LINK VIDEO YOUTUBE

https://youtu.be/UPQW-X_pWcY

Daftar Pustaka

https://repository.unikom.ac.id/37395/1/5-Algoritma%20Greedy.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf