LAPORAN TUGAS BESAR I
IF2211 STRATEGI ALGORITMA
PEMANFAATAN ALGORITMA GREEDY DALAM PEMBUATAN BOT PERMAINAN DIAMONDS

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma pada Semester 2 (dua) Tahun Akademik 2023/2024.

Oleh Kelompok Tulas:

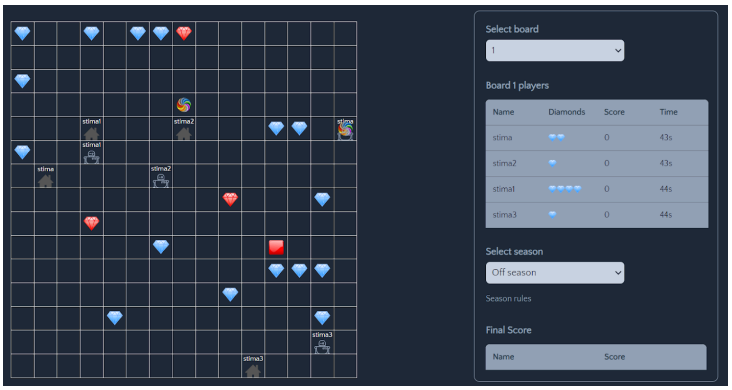
Thea Josephine Halim	13522012
Melati Anggraini	13522035
Hayya Zuhailii Kinasih	13522102

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

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Diamonds merupakan suatu *programming challenge* yang mempertandingkan bot yang dibuat dengan bot dari para pemain lainnya. Sebuah bot memiliki tujuan untuk mengumpulkan *diamond* sebanyak-banyaknya. Pemenang dari permainan ini adalah pemain dengan bot yang memiliki *diamonds* terbanyak. Cara mengumpulkan *diamond* tersebut tidak akan sesederhana itu, tentunya akan terdapat berbagai rintangan yang akan membuat permainan ini menjadi lebih seru dan kompleks. Untuk memenangkan pertandingan, setiap pemain harus mengimplementasikan strategi tertentu pada masing-masing bot-nya.



Gambar 1.1 Tampilan permainan diamond challenge

Dalam permainan *diamond challenge* ini, terdapat beberapa komponen yang perlu diperhatikan:

1. Tipe Diamond

Terdapat 2 jenis *diamond* yaitu *diamond* biru dan *diamond* merah. *Diamond* merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. *Diamonds* akan di-*regenerate* secara berkala dan rasio antara *diamond* merah dan biru ini akan berubah setiap *regeneration*.

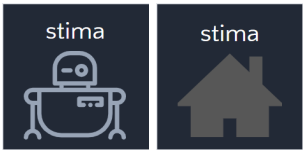
2. Red Button

Red button berperan sebagai *randomizer.* 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 dan Bases



Pada game ini kita akan menggerakkan bot untuk mendapatkan *diamond* sebanyak banyaknya. Semua bot memiliki sebuah *base* yang akan digunakan untuk menyimpan *diamond* yang sedang dibawa dan mengosongkan *inventory* bot. Apabila *diamond* disimpan ke *base, score* bot akan bertambah senilai *diamond* yang dibawa dan *inventory* (akan dijelaskan di bawah) bot menjadi kosong. Perlu dicantumkan juga 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 (selanjutnya akan disebut *tackle*).

5. Bot Inventory Size

Bot memiliki *inventory* yang berfungsi sebagai tempat penyimpanan sementara *diamond* yang telah diambil. *Inventory* ini memiliki kapasitas maksimum sehingga sewaktu waktu bisa penuh. Agar *inventory* ini tidak penuh, bot bisa menyimpan isi *inventory* ke *base* agar *inventory* bisa kosong kembali.

BAB II
TEORI SINGKAT

Dalam Tugas Besar 1 Strategi Algoritma ini, kami diminta untuk membuat bot permainan *diamond challenge* dengan menggunakan algoritma *greedy*.

2.1 Algoritma Greedy

Algoritma *greedy* merupakan metode yang cukup populer untuk menyelesaikan masalah optimasi. Pendekatan ini berfokus pada pengambilan keputusan sekarang yakni memilih opsi yang jelas paling menguntungkan saat ini tanpa mempertimbangkan konsekuensi di masa depan dengan harapan bahwa setiap langkah akan membawa lebih dekat dengan ke solusi akhir yang optimal. Dalam algoritma *greedy,* terdapat enam elemen:

a. Himpunan kandidat: himpunan berisi kandidat yang berpotensi dipilih sebagai solusi
b. Himpunan solusi: himpunan berisi kandidat yang sudah diseleksi sebagai solusi
c. Fungsi solusi (*solution function*): menentukan apabila sudah ditemukan solusi yang optimal
d. Fungsi seleksi (*selection function*): mengambil kandidat terbaik sebagai solusi
e. Fungsi kelayakan (*feasibility function)*: cek apakah kandidat bisa digunakan sebagai solusi
f. Fungsi obyektif (*objective function)*: memaksimumkan atau meminimumkan

Tidak semua solusi yang dihasilkan oleh algoritma *greedy* ini optimal bisa jadi merupakan solusi *sub-optimum* atau *pseudo-optimum*. Diperlukan pembuktian secara matematis untuk mengklaim bahwa solusi yang dihasilkan algoritma *greedy* ini optimum. Salah satu keuntungan penggunaan algoritma *greedy* adalah dalam hal kebutuhan waktu, algoritma ini tidak memakan waktu yang banyak untuk mendapatkan solusi yang eksak.

Terdapat 2 persoalan *greedy* yaitu terkait dengan maksimasi dan minimasi. Contoh-contoh persoalan yang dapat diselesaikan dengan algoritma *greedy* antara lain persoalan penukaran uang (*coin exchange problem*), persoalan memilih aktivitas (*activity selection problem*), minimisasi waktu di dalam sistem, persoalan *knapsack*, penjadwalan *job* dengan tenggat waktu, pohon merentang minimum, lintasan terpendek, kode huffman, pecahan mesir, dll.

2.1.1 Travelling Salesman Problem

2.2 Implementasi Algoritma Greedy Dalam Bot

Proses implementasi dimulai dari analisis masalah, yaitu cara mengumpulkan *diamonds* sebanyak-banyaknya. Persoalan ini kemudian didekomposisi menjadi masalah-masalah kecil. Yang paling penting dan utama untuk diperhatikan adalah strategi *greedy* bot mengambil *diamond* dan kembali ke *base* dengan optimal. Maka dari itu, dengan metode *next_move*, setiap bot akan melakukan pergerakan, akan dilakukan perhitungan berbasis strategi *greedy* langkah ke mana yang paling menguntungkan. Inti dari semua program ini berpusat pada penentuan *goal* atau tujuan yang dituju bot, sehingga *setting goal position* sangatlah krusial.

Pertama-tama akan dilakukan *listing variables* dan *position* objek-objek (*red button, diamonds,* dan *teleporters)*. Posisi setiap jenis objek permainan ini akan disimpan dalam list, secara berurutan sebagai berikut: *red_button_objects, diamond_position,* dan *teleport_objects*. Proses akan dilanjutkan dengan perbandingan jarak posisi sekarang ke base dengan sisa waktu yang ada. Tujuan

pengecekan ini adalah untuk akhir permainan, sehingga *diamond* seadanya yang ada di kantong bot tetap dibawa kembali ke *base* dengan sisa waktu yang ada. Maka dari itu, di bagian awal tujuan akhir (*goal*) akan di set ke posisi base jika kantong sudah penuh (5 diamonds) atau jarak ke base sudah lebih dari sama dengan sisa waktu - 2 (dikurangi 2 karena adanya offset waktu pemrosesan) yang dipastikan distance base > 0 (menghindari *invalid move* karena bot persis berada di atas base dan menyebabkan kedua *delta_x* dan *delta_y* nol).

Jika tujuan bot tidak di-*set* menuju base, bot akan lanjut mencari *diamond* untuk diambil. Di sinilah prinsip algoritma *greedy* ikut andil bagian. Awalnya kami terpikirkan dengan strategi *greedy by density*, *greedy by profit per distance*. Seperti persoalan *knapsack* yang telah diajarkan pada kelas sebelumnya, kami melakukan pemilahan nilai terbesar bobot *diamond* dibagi jarak *diamond* ke posisi sekarang. Akan tetapi, setelah dipikirkan kembali dan berdasarkan analisis dan percobaan, hasil yang diberikan dengan *greedy by density* kurang optimal (mendapat poin sedikit) sehingga kami memutuskan untuk menggunakan *greedy by value/profit.* Tujuan bot akan di-*set* ke arah *diamond* yang terpilih tersebut. Dalam penentuan *diamond* yang dituju, kami juga mempertimbangkan penggunaan komponen *teleporter* dan *red button*. Kami memutuskan untuk *set* tujuan bot ke *red button* apabila tidak ada *diamond* dalam radius 8 kotak, dan jarak bot ke *red button* lebih kecil daripada jarak bot ke diamond terdekat.

Setelah melakukan analisis lebih lanjut, masalah yang muncul apabila *diamonds* di kantong bot adalah 4 (maksimum *inventory* - 1). Kasus ini menjadi masalah karena ada kemungkinan bot akan berusaha mengambil *diamond* merah yang bernilai 2 poin, sehingga bot akan *stuck* dalam status berusaha untuk mengambil diamond tersebut berulang-ulang hingga waktu akan habis dan harus kembali ke base. Maka dari itu, kami membuat kasus terkait hal ini dengan memastikan bot tidak akan melakukan *infinite loop* pada *diamond* merah tersebut. Kami menyeleksi semua kemungkinan *diamond* biru yang paling dekat, dan tentu saja memperhatikan komponen *teleporter*. Lalu *diamond* biru terdekat itu dibandingkan jaraknya dengan jarak posisi sekarang ke *base*. Jika posisi sekarang ke *base* lebih kecil dibandingkan dengan jarak sekarang ke *diamond* ditambah jarak

diamond ke *base*, tujuan akan di-*set* ke base. Perbandingan ini penting untuk mengetahui apakah mengambil 1 *diamond* biru lebih menguntungkan daripada mengosongkan kantong secepatnya.

Pada kasus *tackle bot*, program kami juga dapat melakukan *offensive* dan *defensive mode.* Bot akan melakukan *greedy tackle* menyerang bot lain apabila jarak bot lain persis bersebelahan (1 petak). Sebelumnya program akan melakukan pengecekan apabila *diamond* di *inventory* bot lain yang akan di-*tackle* lebih besar dari diamond di *inventory* bot. Jika ya, bot akan segera melakukan penyerangan dengan melakukan *set* posisi tujuan ke lokasi bot tersebut. Untuk mode *defensive*, bot akan selalu memberi jarak 3 petak terhadap bot lain, berjaga-jaga supaya bot lain tidak bisa melakukan *tackle.* Akan tetapi, perlu diketahui bahwa bot akan masuk ke mode *defensive* apabila diamond di *inventory* bot lain tidak penuh dan lebih kecil dari isi *inventory* bot kita. Terdapat 4 variabel: *right, left, up,* dan *down* yang menjadi penanda apakah bot aman untuk melakukan pergerakan menuju arah tersebut. Variabel tersebut akan di-*set* menjadi *false* apabila terdapat bot lain yang berpotensi melakukan *tackle* pada arah tersebut. Pengecekan dilakukan dengan membandingkan posisi sumbu x dan y kedua bot. Keempat variabel tersebut akan di-*set* ulang setiap pemanggilan fungsi *next_move.* Apabila berdasarkan posisi tujuan bot akan bergerak ke arah dengan salah satu dari keempat variabel bernilai *false*, bot akan bergerak ke arah berlawanan darinya.

Setelah menentukan tujuan akhir bot selanjutnya dengan algoritma *greedy*, program akan melakukan pemanggilan fungsi *get_direction* untuk menentukan nilai *delta_x* dan *delta_y,* penentu langkah bot. Seluruh algoritma pada metode *next_move* akan dipanggil setiap bot selesai melangkah hingga waktu yang ada habis.

2.3 Struktur Folder

Struktur Folder

```
|----- Tubes1_Tulas
          |----- doc
          |----- src


.
|----- src
|           |----- starter-pack
|           |----- game
|           |          |          |----- logic
|           |          |          |          |----- coba.py
|           |          |          |          |----- base.py
|           |          |          |----- api.py
|           |          |          |----- board_handler.py
|           |          |          |----- bot_handler.py
|           |          |          |----- models.py
|           |          |          |----- util,py
|           |          |----- decode.py
|           |          |----- main.py
```

Program akan di-*run* melalui file run-bots.bat yang dijalankan dari direktori src/tubes1-IF2211-bot-starter-pack-1.0.1/game dengan package yang di-*load* sebagai berikut :

- main.py
- packages backend (direktori game-engine)
- packages frontend (direktori game-engine)
- logic/nama_bot.py

Dengan meninjau struktur folder diatas, secara garis besar terdapat 2 program besar yang akan dijalankan yaitu *game engine* yang berisi kode frontend dan backend yang memberi tempat/ruang dijalankanya bot serta *bot-starter-pack* yang akan memanggil API backend, logika bot, dan program utama. Di akhir permainan, akan ditampilkan board skor hasil permainan.

2.4 Cara Menjalankan Program

Cara menjalankan game :

1. Install semua *requirements* yang dibutuhkan atau lebih detailnya bisa baca README pada github terlampir.
2. Jalankan *game-engine*

3. Jalankan *bot-starterpack*
4. Jika ingin melakukan run pada satu bot, pada root path command di bawah ini

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

Sedangkan untuk menjalankan beberapa bot jalankan:

- Untuk windows

```
./run-bots.bat
```

- Untuk Linux/ macOS

```
/run-bots.sh
```

Dengan melakukan penyesuaian script yang ada pada *run-bots.bat* atau *run-bots.sh* dari segi logic yang digunakan, email, nama, dan password.

BAB III
APLIKASI GREEDY ALGORITHM

3.1 Pemetaan Bot ke Elemen-Elemen Algoritma Greedy

Dalam permainan *Diamond*, seorang pemain dapat melakukan enam aksi diantaranya bergerak (maju, mundur, kanan, kiri), menggunakan *teleporter*, mengejar bot, menggunakan *red button*, menghindari serangan bot lawan dan kembali ke *base*. Semua aksi tersebut dimasukkan ke dalam himpunan kandidat secara implisit melalui aksi-aksi bot dengan pengecualian yakni kembali ke *base*.

Selanjutnya, untuk fungsi solusi, fungsi seleksi, fungsi kelayakan, dan fungsi obyektif tidak dapat diimplementasikan secara langsung dalam proses pembuatan algoritma *greedy* karena permasalahan yang diselesaikan adalah kompetisi antara bot pemain yang strateginya berbeda-beda. Namun, secara implisit kita dapat memperkirakan fungsi objektif yang bisa membantu bot memperoleh skor terbanyak. Maka dari itu, terciptalah fungsi solusi yaitu memperbesar peluang kemenangan bot yang ditandingkan.

Implementasi fungsi seleksi dilakukan melalui analisis kondisi bot yang bermain saat itu serta menggunakan faktor-faktor yang lain seperti ukuran inventory, jarak bot dengan diamond, jarak ke base, waktu tersisa, jarak bot dengan bot lain, dan jumlah *diamonds* di suatu area. Pengambilan keputusan bot didasarkan pada keuntungan terbesar yang akan didapat (*greedy*) saat itu. Berikut analisis-analisis kondisi, aksi, dan fungsi yang digunakan.

No.	Kondisi	Aksi	Fungsi Utama
1.	Isi *inventory* pemain < maksimum *inventory* - 1 dan aman dari bot lain dalam jarak 3 petak ke depan	Set *goal_position* ke diamond teroptimal tersebut	*closest_diamond*
2.	Jarak *teleporter*	Gunakan *teleporter*	*get_teleport_objects,*

	pertama dari posisi sekarang + jarak *teleporter* kedua ke posisi tujuan < jarak tempuh normal	pertama (dan sebaliknya untuk *teleporter* kedua)	*closest_diamond, closest_to_position*
3.	Isi *inventory* pemain terisi 4 *diamonds* (maksimum *inventory* -1) dan *diamond* terdekat adalah *diamond* merah.	● Jika terdapat *diamond* biru, jika jarak ke *diamond* kurang dari jarak ke base maka menuju *diamond*. Jika sebaliknya maka menuju ke *base*. ● Jika tidak ada *diamond* biru maka ke *base*.	*closest_diamond, closest_to_position*
4.	Isi *inventory* pemain sudah terisi penuh	Kembali ke *base*	*get_teleport_objects, closest_to_position*
5.	Waktu yang tersisa - 2 sama dengan waktu untuk menempuh perjalanan kembali ke base (asumsi 1 petak butuh 1 detik)	Kembali ke *base*	
6.	Terdapat bot lain pada jarak 1 petak (persis bersebelahan) dan isi *inventory* lebih banyak	Segera *tackle* dengan set *goal_position* dengan lokasi bot	

7.	Terdapat bot lain pada jarak 3 petak dan isi *inventory* lebih sedikit yang berpotensi melakukan *tackle*	Menghindar dengan bergerak berlawanan arah datang bot lain	*avoid_tele_base, safe*
8.	Jika waktu tersisa dikurang 2 sama dengan jarak ke base	Kembali ke base	
9.	Jika bot diam (*invalid move).*	Jika bot di set ke *teleporter* sedangkan posisi bot sekarang di *teleporter*.	
10.	Tidak ada *diamond* di area bot radius 7 meter dan jarak menuju *diamond* lain terdekat > jarak menuju *red button*	Bot set *goal_position* ke *red button*	*get_red_button_objects*

Tabel 3.3.1 Analisis kondisi, aksi, dan fungsi

3.2 Eksplorasi Solusi Alternatif

Terdapat beberapa alternatif solusi yang pernah kita implementasikan akan tetapi hasilnya kurang memuaskan untuk berbagai pertimbangan. Berikut penjelasannya:

a. *Greedy by area*

Algoritma Greedy diimplementasikan dengan mencari area-area yang memiliki diamond banyak. Kita membagi area-area tersebut menjadi 8 bagian. Mungkin di beberapa kondisi, seperti ketika area-area tersebut tidak ada diamond, akan memberi keuntungan yang lebih baik, akan tetapi dalam banyak kondisi lain, implementasi algoritma ini dinilai kurang efektif

karena hanya akan menghabiskan waktu bot untuk berjalan menuju ke area tersebut dan bot mengabaikan diamond-diamond di sekitarnya ketika ia berjalan menuju area tersebut padahal jaraknya lebih dekat. Oleh karena itu, dengan pertimbangan keuntungan skor yang akan didapat, maka kami memilih untuk tidak mengimplementasikan di bot kami. *Greedy by area* akan menguntungkan hanya jika secara kebetulan base berada di area dengan diamonds yang banyak.

b. *Greedy by density*

Algoritma ini dibuat dengan memperhatikan 2 faktor yaitu jarak dan bobot diamond. Density yang dimaksud disini adalah perbandingan nilai bobot diamond dibagi dengan jarak menuju diamond tersebut. Posisi akan diubah(*goal_position)* menuju kondisi yang memberikan nilai perbandingan lebih besar. Gambaran kasus dalam board bot untuk implementasi algoritma ini adalah ketika bot berada di posisi yang jarak merah dan biru sama, sehingga jika kita menggunakan greedy, kita akan memilih menuju ke diamond merah yang memberikan skor lebih banyak. Akan tetapi, berdasarkan beberapa analisis percobaan, *greedy by density* memberikan hasil akhir yang lebih sedikit daripada *greedy by profit*. Oleh karena itu, kami memasukkan solusi ini ke solusi alternatif.

c. *Extreme tackle*

Awalnya kami terpikirkan untuk menyerang bot lawan dengan jarak antar bot yang lebih jauh tujuannya agar frekuensi penyerangan lebih banyak. Dengan peningkatan frekuensi ini kami berharap agar inventory bot kami dapat langsung terisi penuh / hampir penuh dengan usaha minimal atau tidak perlu repot mencari diamond-diamond di sekitar cukup ambil dari bot lain saja sembari mengganggu proses pengambilan diamond bot lain. Namun, setelah dilakukan eksplorasi dan analisis lebih jauh, *extreme tackle* dapat menjadi tidak efektif apabila bot lain memiliki algoritma *defense* (menghindari bot lain sembari mencari diamond lain atau kembali ke base) sehingga bot akan terus-menerus mengejar bot lain. Ada kemungkinan juga bot yang dikejar memiliki diamond di *inventory* yang

lebih sedikit (*not worth*), atau malah bot yang dikejar berhasil mencapai base sehingga bot kami akan membuang waktu saja untuk pengejaran tanpa membawa keuntungan apa-apa.

Strategi *extreme tackle* dapat menjadi riskan juga apabila kita berhasil mengejar bot lain, tetapi justru bot lain memiliki algoritma penyerangan tiba-tiba yang menyebabkan bot kita sendiri yang terkena *tackle.* Kecepatan algoritma menjadi penting dalam kecepatan gerak bot, sehingga kita tidak bisa menilai apakah bot kita jauh lebih cepat dari bot lain, atau kebetulan-kebetulan yang lain.

3.3 Analisis Solusi

a. *Basic movement* menuju diamond teroptimal

Program menggunakan algoritma greedy untuk melakukan analisis diamond teroptimal, yaitu diamond terdekat entah dengan menggunakan sistem *teleporter* ataupun dengan pergerakan biasa. Implementasi program memanfaatkan metode *closest_diamond* sebagai penentu diamond yang dituju dan metode teroptimal untuk mencapai tujuan tersebut.

b. Penggunaan *teleporter*

Objek *teleporter* menjadi salah satu kunci penting dalam algoritma greedy ini. *Teleporter* bisa mengurangi jumlah langkah yang diperlukan bot baik dalam proses pengambilan diamond maupun kembali menuju base. Program akan melakukan kalkulasi perbandingan pergerakan biasa tanpa *teleporter* dengan mengakses *teleporter* sebagai *shortcut.* Perhitungan penggunaan *teleporter* ini juga mempertimbangkan jarak yang diperlukan untuk menuju objek *teleport* A ditambah dengan jarak yang diperlukan dari objek *teleport* B menuju *goal position* yang dituju oleh bot. Implementasi objek *teleporter* ini terdapat pada fungsi *closest_diamond* dan *closest_to_position.*

c. Mengejar bot (*tackle*)

Konsep tackle pada program kami tidak menjadi perhatian utama, sebab kami rasa *offense* justru dapat membahayakan jika kita tidak mengetahui algoritma bot lain. Bisa saja kita melakukan pengejaran, dan berhasil *catch up* dengan bot lain, tetapi justru bot lain melakukan set up penyerangan tiba-tiba yang berimbas pada kekalahan diri dan kehilangan waktu untuk proses pengejaran tadi. Walaupun begitu, kami memutuskan untuk melakukan implementasi *tackle* hanya ketika jarak bot dengan bot lain adalah 1, atau bersebelahan persis, dan isi *inventory* bot lain lebih dari *inventory* bot kami. Hal ini bertujuan untuk menangani kasus apabila bot lain berhasil *catch up,* sehingga diperlukan pergerakan cepat untuk *tackle* sebaliknya.

d. Menggunakan *red button*

Objek *red button* pada program kami akan difungsikan ketika kondisi jumlah diamond di sekitar bot tidak ada di area tertentu yaitu dengan jarak lebih dari 7 petak dan jarak ke diamond lebih dari jarak ke *red button,* yang artinya *red button* berada di area kosong tersebut. Ukuran jarak 7 kami putuskan dengan memperhatikan ukuran petak keseluruhan yaitu 15x15 (mengambil peluang luas yang paling menguntungkan). Meskipun persebaran diamond red button akan teracak secara random nantinya, tapi setelah dipikirkan dan dicoba kembali ternyata dengan kemunculan diamond-diamond kembali akan lebih menguntungkan untuk meningkatkan skor daripada harus mengejar diamond yang terlampau jauh.

e. Menghindari bot lawan (*defensive*)

Bot di-*set* untuk menghindari lawan berjarak 3 petak dan memiliki diamond di *inventory* lebih sedikit dari *inventory* kami, untuk mencegah *tackle*.

f. Kembali ke base

Bot melakukan pergerakan kembali ke base apabila terjadi 3 kasus:

1) Kalkulasi waktu tersisa - 2 sama dengan jarak dari lokasi bot ke base.
2) Jumlah diamond di *inventory* mencapai maksimum, atau pada *default*-nya 5.
3) Jumlah diamond di *inventory* 4, dan jarak ke base jauh lebih dekat daripada jarak yang harus ditempuh untuk mengambil ekstra 1 diamond biru lagi.

3.4 Strategi yang Terpilih

Berdasarkan pertimbangan dan analisis solusi-solusi alternatif, logic bot akan menggunakan *greedy by profit* dan *distance* yang jauh lebih optimal secara waktu dan kinerja untuk mendapatkan banyak diamonds. Pengisian *inventory* akan dilakukan dengan pemanfaatan fungsi *closest_diamond* dan *closest_to_position.* Dengan *greedy by profit and distance,* program akan menentukan diamond yang paling dekat dengan bot dengan berbagai pertimbangan seperti penggunaan teleport. *Greedy by profit and distance* yang diaplikasikan juga dilengkapi dengan pertimbangan untuk menggunakan red button apabila jarak ke diamond teroptimal sudah terlalu jauh. Selain itu, juga terdapat pertimbangan menggunakan *use_teleport* jika jarak menuju diamond atau ke base lebih bisa lebih dekat dengan menggunakan teleport.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Pseudocode Program

4.1.1 Function *avoid_tele_base*

```
function avoid_tele_base(current_position: Position, up, down,
left, right, board: Board, delta_x, delta_y) ->
(delta_x,delta_y)
{Langkah menghindari bot dengan bergerak ke arah berlawanan}

KAMUS
-

ALGORITMA
if(safe(delta_x, delta_y, up, down, left, right) = False) then
   if bot ada di sumbu x then
      if posisi terakhir>= tinggi bot/2 then
         -> 0,1 (up)
      else
        -> 0,-1(down)
   else
     If current_position->x >= lebar/2 then
        -> -1,0 (left)
     else
       -> 1,0 (right)
else
   -> 0,0
```

4.1.2 Function *safe*

```
function safe(delta_x,delta_y,up,down,left,right) -> bool
{Mengecek apakah aman bergerak ke atas, bawah, kanan, atau kiri}

KAMUS
-

ALGORITMA
if ((delta_x == 1) and right) or ((delta_x == -1) and left) or
((delta_y == 1) and up) or ((delta_y == -1) and down) then
        -> True
-> False
```

4.1.3 Function *get_teleport_objects*

```
function get_teleport_objects(self, board: Board) -> array of
GameObjects
{Mengembalikan objek game bertipe teleport}

KAMUS
-

ALGORITMA
-> [d for d in board.game_objects if d.type ==
"TeleportGameObject"]
```

4.1.4 Function *get_red_button_objects*

```
function get_red_button_objects(self, board: Board) -> array of
GameObjects
{Mengembalikan objek game bertipe red/diamond button}

KAMUS
-

ALGORITMA
-> [d for d in board.game_objects if d.type ==
"DiamondButtonGameObject"]
```

4.1.5 Function *get_bot_objects*

```
function get_bot_objects(self, board: Board) -> array of
GameObject
{Mengembalikan objek game bertipe bot}

KAMUS
-

ALGORITMA
-> [d for d in board.game_objects if d.type == "BotGameObject"]
```

4.1.6 Function *closest_diamond*

```
function closest_diamond(self, board: Board, board_bot:
GameObject, diamonds: List[GameObject])-> dict[str, any]
{Mengembalikan dictionary berisi diamond teroptimal, metode, dan
jarak}

KAMUS
current_position : Position
min, method, idx, this_method, dist_teleporter_1,
dist_teleporter_2 : integer
teleporters : array of GameObject

ALGORITMA
current_position <- board_bot.position
min <- 9999
method <- 0
idx <- -1
```

```
teleporters <- self.get_teleport_objects(board)
i traversal (0..len(diamonds))
    dist <- abs(current_position.x - diamonds[i].position.x) +
    abs(current_position.y - diamonds[i].position.y)
    this_method <- 0
    dist_teleporter_1 <- abs(current_position.x -
    teleporters[0].position.x) + abs(current_position.y -
    teleporters[0].position.y) + abs(diamonds[i].position.x -
    teleporters[1].position.x) + abs(diamonds[i].position.y -
    teleporters[1].position.y)
    dist_teleporter_2 <- abs(current_position.x -
    teleporters[1].position.x) + abs(current_position.y -
    teleporters[1].position.y) + abs(diamonds[i].position.x -
    teleporters[0].position.x) + abs(diamonds[i].position.y -
    teleporters[0].position.y)
    if (dist_teleporter_1 <= dist) and (dist_teleporter_1 <=
dist_teleporter_2) then
        dist <- dist_teleporter_1
        this_method <- 1
    elif (dist_teleporter_2 < dist) and (dist_teleporter_2 <
dist_teleporter_1) then
        dist <- dist_teleporter_2
        this_method <- 2
    if (dist < min) then
        min <- dist
        method <- this_method
        idx <- i
-> {"diamond": diamonds[idx], "method": method, "distance": min}
```

4.1.7 Function *next_move*

```
function next_move(self, board_bot: GameObject, board: Board) ->
(int, int) {Menentukan pergerakan selanjutnya}

KAMUS
teleport_objects, red_button_objects, bot_objects,
diamond_list,bot_position, bot_pockets, distance_bot_list,
blue_diamond_list : array
base, current_position : Position
props : Properties
time_left, my_bot_pocket, idx_our_bot, delta_x, delta_y : int
avoid, left, right, down, up : boolean
to_base, to_red_button : dict[str, any]
goal_diamond :dict[str, any]


ALGORITMA
{state of board}
teleport_objects <- self.get_teleport_objects(board)
red_button_objects <- self.get_red_button_objects(board)
bot_objects <- self.get_bot_objects(board)
diamond_list <- board.diamonds
base <- board_bot.properties.base
to_base <- self.closest_to_position(board, board_bot, base)
time_left <- board_bot.properties.milliseconds_left/1000
to_red_button <- self.closest_to_position(board, board_bot,
red_button_objects[0].position)

{state of bot}
props <- board_bot.properties
current_position <- board_bot.position
```

```
my_bot_pocket <- board_bot.properties.diamonds

{base variables for tackle}
avoid <- false
left <- true
right <- true
up <- true
down <- true

if props.diamonds = 5 or (to_base["distance"] >= time_left-2 and
to_base["distance"]>0) then {go back to base}
       self.goal_position <- base
       self.travel_method <- to_base["method"]

else:
       bot_position <- [i.position for i in bot_objects]
       bot_pockets <- [j.properties.diamonds for j in
       bot_objects]

{attack other bots if distance is = 1 and they have more
diamonds}
distance_bot_list <-  [(abs(current_position.x - position.x) +
abs(current_position.y - position.y)) for position in
bot_position]
i traversal [0, len(bot_objects)]
       if (current_position.x = bot_objects[i].position.x and
       current_position.y = bot_objects[i].position.y) then
             idx_our_bot <- i
       i <- 0
for bot in bot_position
       if(current_position != bot and distance_bot_list[i] = 1
       and bot_pockets[i]>my_bot_pocket) then
             self.goal_position <- bot_position[i]
             delta_x, delta_y <-
       get_direction(current_position.x, current_position.y,
       self.goal_position.x, self.goal_position.y)
             -> delta_x, delta_y {immediately decide to attack}
       i+=1

i traversal [0, len(bot_objects)]:
       if(props.diamonds > bot_objects[i].properties.diamonds
       and bot_objects[i].properties.diamonds!=5 and
       i!=idx_our_bot and distance_bot_list[i]=3) then
             {determine position of threat}
             if (bot_objects[i].position.x >
             board_bot.position.x) and
             (bot_objects[i].position.y > board_bot.position.y)
             then
                   up,right <- False,False
             else if (bot_objects[i].position.x >
             board_bot.position.x) and
             (bot_objects[i].position.y < board_bot.position.y)
             then
                   down,right <- False,False
             else if (bot_objects[i].position.x <
             board_bot.position.x) and
             (bot_objects[i].position.y < board_bot.position.y)
             then
                   left,down <- False,False
             else if (bot_objects[i].position.x <
             board_bot.position.x) and
             (bot_objects[i].position.y > board_bot.position.y)
             then
```

```
                left,up <- False,False
            else if (bot_objects[i].position.x =
            board_bot.position.x) then
                  if (bot_objects[i].position.y >
                  board_bot.position.y) then
                        up <- False
                  else
                        down <- False
            else if (bot_objects[i].position.y =
            board_bot.position.y) then
                  if (bot_objects[i].position.x >
                  board_bot.position.x) then
                        right <- False
                  else
                        left <- False

            avoid <- True

{get the closest diamond}
goal_diamond <- self.closest_diamond(board, board_bot,
diamond_list)
self.goal_position <- goal_diamond["diamond"].position
self.travel_method <- goal_diamond["method"]

if (props.diamonds = 4 and
goal_diamond["diamond"].properties.points = 2) then
      blue_diamond_list <- list(filter(lambda diamond:
      diamond.properties.points = 1, diamond_list))
      if (len(blue_diamond_list) > 0) then
            goal_diamond <- self.closest_diamond(board,
      board_bot, blue_diamond_list)
            if(goal_diamond["distance"] < to_base["distance"])
            then {diamond is closer}
                  self.goal_position <-
                  goal_diamond["diamond"].position
                  self.travel_method <- goal_diamond["method"]
            else {base is closer}
                  self.goal_position <- base
                  self.travel_method <- to_base["method"]
     else
            self.goal_position <- base
            self.travel_method <- to_base["method"]

{go to red button if diamond is scarce}
if (to_red_button["distance"]<goal_diamond["distance"] and
goal_diamond["distance"]>7) then
      self.goal_position <- red_button_objects[0].position
      self.travel_method <- to_red_button["method"]

if self.goal_position then
      {change goal to teleporter}
      if (self.travel_method = 1) then
            self.goal_position <- teleport_objects[0].position
      else if (self.travel_method = 2) then
            self.goal_position <- teleport_objects[1].position

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

      if avoid then
            delta_x, delta_y <-
            avoid_tele_base(current_position,up,down,left,right
            ,board,delta_x,delta_y)
```

```
       while not board.is_valid_move(current_position, delta_x,
 delta_y) do
            random_direction <- random.choice([(1, 0), (0, 1),
 (-1, 0), (0, -1)])
            delta_x <- random_direction[0]
            delta_y <- random_direction[1]

      -> delta_x, delta_y
```

4.1.8 Function *closest_to_position*

```
function closest_to_position(self, board: Board, board_bot:
GameObject, goal: Position) -> dict[str,any]
{Menentukan metode untuk mencapai suatu lokasi tujuan}

KAMUS
current_position : Position
teleporters : array of GameObject
method, dist, dist_teleporter_1, dist_teleporter_2 : integer

ALGORITMA
current_position <- board_bot.position
teleporters <- self.get_teleport_objects(board)
method <- 0
dist <- abs(current_position.x - goal.x) + abs(current_position.y
- goal.y)
dist_teleporter_1 <- abs(current_position.x -
teleporters[0].position.x) + abs(current_position.y -
teleporters[0].position.y) + abs(goal.x -
teleporters[1].position.x) + abs(goal.y -
teleporters[1].position.y)
dist_teleporter_2 <- abs(current_position.x -
teleporters[1].position.x) + abs(current_position.y -
teleporters[1].position.y) + abs(goal.x -
teleporters[0].position.x) + abs(goal.y -
teleporters[0].position.y)
if (dist_teleporter_1 <= dist) and (dist_teleporter_1 <=
dist_teleporter_2) then
    dist <- dist_teleporter_1
    method <- 1
else if (dist_teleporter_2 < dist) and (dist_teleporter_2 <
dist_teleporter_1) then
    dist <- dist_teleporter_2
    method <- 2
->{"method": method, "distance": dist}
```

4.2 Struktur Data

No.	Struktur Data	Atribut/Method	Deskripsi
1	Bot	name	Nama bot (string)
		email	Email bot (string)
		id	Nomor ID bot (int)

2	Position	y	Posisi vertikal bot (int)
		x	Posisi horizontal bot (int)
3	Properties	points	Bersifat opsional. Jumlah poin sebuah diamond. (int)
		pair_id	Bersifat opsional. Menandakan pasangan teleporter. (str)
		diamonds	Bersifat opsional. Jumlah diamond yang dimiliki suatu bot. (int)
		score	Bersifat opsional. Skor yang didapatkan suatu bot. (int)
		name	Bersifat opsional. Nama bot. (str)
		inventory_size	Bersifat opsional. Ukuran inventory suatu bot. (int)
		can_tackle	Bersifat opsional. Menentukan bot bisa tackle atau tidak. (bool)
		milliseconds_left	Bersifat opsional. Waktu yang tersisa dalam millisecond. (int)
		time_joined	Bersifat opsional. Waktu bot join permainan. (str)
		base	Bersifat opsional. Posisi base suatu bot. (Base)
4	GameObject	id	ID suatu objek. (int)
		position	Posisi suatu objek. (Position)
		type	Jenis suatu objek. (str)
		properties	Bersifat opsional. Properti yang dimiliki suatu objek. (Properties)
5	Board	id	ID board. (int)
		width	Lebar board. (int)

		height	Tinggi board. (int)
		features	List fitur suatu board. (List[Feature])
		minimum_delay_between_moves	Waktu jeda minimum antara setiap gerakan. (int)
		game_objects	Bersifat opsional. List semua objek dalam board, termasuk bot, diamonds, teleporter, dan red button. (List[GameObject])
		bots(self) -> List[GameObject]	Fungsi mengambil list semua bot dalam board.
		diamonds(self) -> List[GameObject]	Fungsi mengambil list semua diamond dalam board.
		get_bot(self, bot: Bot) -> Optional[GameObject]	Fungsi mengambil bot spesifik.
		is_valid_move(self, current_position: Position, delta_x: int, delta_y: int) -> bool	Fungsi menentukan apakah suatu pergerakan valid atau tidak.

Tabel 4.2.1 Struktur Data

4.3 Pengujian

Yang diuji	Implementasi Strategi	Hasil
Mengambil diamond	Strategi diimplementasikan dengan mengambil diamond yang terdekat.	Bot berhasil mengambil diamond terdekat. Namun, terdapat kejadian bot tidak sengaja masuk ke petak teleporter ketika menuju diamond dan berakhir jauh dari base.
Men-*tackle* bot lawan	Strategi diimplementasikan dengan men-*tackle* ketika bot lawan berada persis di samping bot dan bot lawan memiliki lebih banyak diamond.	Dalam beberapa situasi, bot berhasil men-*tackle* bot lawan. Namun, terdapat hasil yang tidak optimal yaitu ketika bot terjebak mengejar bot lawan dan ketika bot dan

		bot lawan saling men-*tackle* di waktu yang hampir bersamaan.
Menghindar bot lawan	Strategi diimplementasikan dengan berpindah arah ketika terdapat bot lawan yang dekat dan bot memiliki lebih sedikit diamond.	Strategi sudah memberikan hasil yang optimal.
Menuju ke base	Strategi diimplementasikan dengan pergi ke base ketika sudah tidak dapat mengambil diamond lagi atau waktu akan habis.	Dalam mayoritas kasus, bot berhasil menuju ke base dengan optimal. Namun, bisa terjadi bot tidak sengaja terkena teleporter ketika menuju ke base, sehingga bot harus membuang waktu masuk ke teleporter lagi untuk meneruskan perjalanan ke base. Terdapat kasus khusus ketika bot ingin ke base ketika waktu akan habis, yaitu bot tidak sempat ke base.
Menggunakan teleporter	Strategi diimplementasikan dengan memilih untuk menggunakan teleporter ketika akan membawa ke tujuan dengan lebih cepat.	Strategi sudah memberikan hasil yang optimal. Mungkin saja teleporter berpindah tempat sebelum bot sampai ke teleporter, namun karena teleporter biasanya digunakan ketika bot dekat dengan teleporter, maka sedikit waktu yang dihabiskan untuk berpindah arah.
Menggunakan red button	Strategi diimplementasikan dengan menuju ke red button apabila diamond	Strategi sudah memberikan hasil yang optimal.

	yang tersisa lebih jauh daripada red button.	

Tabel 4.3.1 Hasil Pengujian

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Tugas besar 1 IF2211 Strategi Algoritma mengungkit masalah mengenai pemilihan strategi algoritma greedy sebagai pemecah permainan *diamond challenge*. Diperlukan pencarian implementasi algoritma greedy yang tepat agar bot dapat memperoleh poin sebanyak-banyaknya sembari memanfaatkan komponen-komponen yang tersedia, seperti perbedaan tipe diamond, skema *tackle, teleporters*, dan *red button*. Algoritma berbasis *greedy by distance and profit* terpilih sebagai algoritma terbaik dalam kasus ini dengan berbagai penambahan fungsi-fungsi penanganan kasus. Kasus-kasus yang penting untuk ditangani adalah kembali ke base di waktu-waktu terakhir, *defense mode* menghindari bot lain, *tackle* bot, dan penggunaan red button.

5.2 Saran

Selama pengerjaan tugas besar ini kami memiliki beberapa saran untuk pengerjaan tugas besar berikutnya, di antaranya:

- Membuat sheets berisi link-link terintegrasi yang dibutuhkan selama pengerjaan tugas
- Mencari referensi lebih banyak lagi sebagai bahan perbandingan
- Melakukan testing dan menandingkan bot dengan bot-bot lain.

LAMPIRAN

Repository : **https://github.com/pandaandsushi/Tubes1_Tulas**

Video : **https://youtu.be/3o7Z10iziZU?si=pHsvXPEEoN2CsAGQ**

DAFTAR PUSTAKA

Alabi, Tantoluwa. (2023). What is a Greedy Algorithm? Examples of Greedy Algorithms. Diakses pada 3 Maret 2024 dari https://www.freecodecamp.org

Asisten, S. A. (2024). Tugas Besar 1 IF2211 Strategi Algoritma 2024. docx. Diakses dari Google Dokumen: https://docs.google.com/document

Engati. (2021). Diakses pada 3 Maret 2024 dari https://www.engati.com/glossary/greedy-algorithm

Munir, R. (2024). Strategi Algoritma #1 Algoritma Greedy (Bagian 1). Diakses dari Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi

Munir, R. (2024). Strategi Algoritma #2 Algoritma Greedy (Bagian 2). Diakses dari Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi

Munir, R. (2023). Strategi Algoritma #3 Algoritma Greedy (Bagian 3). Diakses dari Homepage Rinaldi Munir: https://informatika.stei.itb.ac.id/~rinaldi

Pedamkar, Priya. (2023). What is a Greedy Algorithm? Diakses pada 3 Maret 2024 dari https://www.educba.com/what-is-a-greedy-algorithm/