Tugas Besar 1 IF2211 Strategi Algoritma

Semester II tahun 2023/2024

Pemanfaatan Algoritma Greedy dalam Pembuatan *Bot* Permainan Diamonds

Disusun Oleh:

Ariel Herfrison	13522002
Irfan Sidiq Permana	13522007
Bryan Cornelius Lauwrence	13522033

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2024

BAB 1
DESKRIPSI TUGAS

Permainan “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

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.

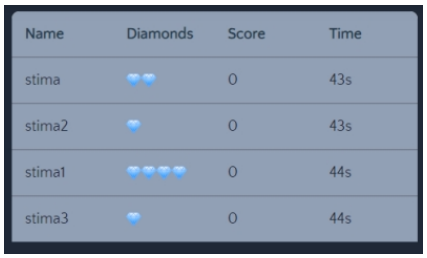
3. Teleporters

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 dan 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.

BAB 2
LANDASAN TEORI

A. Algoritma Greedy

Algoritma greedy adalah salah satu metode alternatif dalam memecahkan masalah optimasi. Karakteristik unik yang membedakan algoritma greedy dengan algoritma lain adalah bahwa algoritma greedy akan mengambil pilihan optimal secara lokal atau pada saat itu juga. Algoritma greedy tidak memperhatikan konsekuensi ke depan, maupun pilihan sebelumnya. Dengan memilih optimum lokal pada setiap langkah, algoritma greedy berharap solusi akhir akan berupa optimum global. Algoritma greedy secara umum dapat diuraikan menjadi elemen-elemen berikut:

1. Himpunan kandidat

   Himpunan kandidat adalah himpunan yang berisi pilihan/kandidat yang dapat dipilih pada setiap langkah

2. Himpunan solusi

   Himpunan solusi adalah himpunan yang merekam pilihan-pilihan yang telah diambil

3. Fungsi solusi

   Fungsi solusi adalah fungsi yang digunakan untuk memeriksa apakah himpunan kandidat sudah memberikan solusi

4. Fungsi seleksi

   Fungsi seleksi adalah fungsi yang digunakan sebagai basis pemilihan kandidat di dalam himpunan kandidat. Fungsi seleksi dirumuskan berdasarkan strategi greedy dan dapat bersifat heuristik.

5. Fungsi kelayakan (feasible)

   Fungsi kelayakan adalah fungsi yang digunakan untuk memeriksa apakah kandidat dapat dapat dipilih berdasarkan syarat persoalan.

6. Fungsi objektif

   Fungsi objektif dalam persoalan optimasi hanya terdapat dua macam, yaitu memaksimumkan atau meminimumkan. Fungsi obyektif akan mempengaruhi arah pemilihan dalam fungsi seleksi.

B. Cara Kerja Permainan

Permainan “Diamonds” terdiri atas dua modul, yaitu Game Engine dan Bot Starter Pack. Game Engine menyimpan kode backend dan frontend permainan. Bot Starter Pack menyimpan program untuk memanggil API yang tersedia pada backend, program utama (main) permainan, dan, yang terutama, program bot logic. Bot dapat diimplementasikan dengan pertama-tama membuat kode baru dengan nama “mybot” dalam direktori “/game/logic”. Lalu, pada kode tersebut, perlu dibuat kelas yang meng-inherit kelas BaseLogic. Pada kelas tersebut, selain mengimplementasikan constructor, perlu juga dibuat method “next_move” yang akan digunakan untuk memanipulasi gerakan bot. Terakhir, kelas yang telah dibuat harus diimport pada main.py dan didaftarkan pada dictionary “CONTROLLERS”. Setelah itu, logika bot dapat diimplementasikan dengan menuliskannya pada method “next_move”. Method “next_move” dapat dimanipulasi sehingga mengembalikan (return) gerakan yang ingin bot lakukan (delta x, delta y). Gerakan tersebut akan dikembalikan ke program main dan akan dijalankan oleh permainkan.

C. Struktur Program

Dalam program Diamonds yang menjadi subjek tugas besar saat ini, terdapat beberapa komponen yang telah disediakan untuk menjalankan permainan Diamonds ini. Komponen-komponen tersebut adalah:

1. *Bot Starter Pack*

   Terdiri dari file-file dalam bahasa pemrograman Python. Pada *starter pack* program, *source code* dibuat dengan paradigma OOP yang berisi kelas-kelas untuk merealisasikan objek yang ada, yaitu bot, *diamonds*, *teleport*, *red button*, *base*, map, dan lain-lain. Selain itu, terdapat file API untuk menghubungkan seluruh program dengan *server*.

2. *Game engine*

   Terdiri dari file untuk menjalankan website dan Docker. *Game engine* juga berfungsi menampilkan seluruh objek yang ada di map.

Untuk bisa mengimplementasikan pergerakan bot, program dibuat melalui file Python yang berada di *bot starter pack*. Penghubung bot dengan *server*, beserta seluruh pergerakanannya, dilakukan pada file main.py. Pada folder *game* terdapat file models.py yang berfungsi untuk merealisasikan kelas-kelas untuk objek pada permainan dan terdapat file util.py yang berisi fungsi untuk menjalankan bot dan membawa bot ke suatu tujuan di dalam permainan. Untuk membuat logika permainan bot, file dibuat di folder *logic*. Untuk membuatnya dapat membuat file baru yang akan diisi dengan seluruh strategi *greedy* menggunakan model pada models.py maupun fungsi-fungsi pada util.py.

D. Alur Jalan Program

Untuk menjalankan permainan serta bot pemain, terdapat beberapa *tools* yang diperlukan yaitu Node.js, Docker Desktop, dan Yarn. Node.js digunakan sebagai *backend* untuk menghubungkan permainan ke *server*, Docker Desktop digunakan untuk menyediakan *environment* untuk menjalankan permainan yang telah di-*pack* menggunakan Docker sebelumnya, serta Yarn dilakukan untuk menginstall semua dependensi yang diperlukan untuk menjalankan permainan. Bila ketiga *tools* tersebut sudah di-install, maka instruksi '*yarn'* dapat dijalankan pada terminal di folder direktori *game engine* untuk menginstall dependensi, lalu melakukan *setup* database lokal menggunakan Docker diikuti dengan menjalankan setup database Prisma yang telah diinstall menggunakan Yarn sebelumnya. Langkah terakhir yaitu menjalankan instruksi '*npm run build*' dan '*npm run start*' untuk menjalankan permainan pada *browser*. Untuk menjalankan bot, file "run-bots.bat" pada folder bot-starter-pack dapat dijalankan pada terminal untuk memunculkan bot pada papan permainan. Setelah *setup* permainan selesai, tiap kali ingin menjalankan permainan langkah yang perlu dilakukan adalah membuka aplikasi Docker Desktop, menjalankan command '*npm run build*' dan '*npm run start*' pada folder game engine, dan menjalankan file 'run-bots.bat' untuk memunculkan bot pada permainan.

E. Alur Pengembangan Strategi

Strategi dibuat dengan mempertimbangkan seluruh objek yang ada di permainan. Karena itulah, diperlukan kelas untuk mendefinisikan objek lebih lanjut beserta fungsi-fungsi baru untuk menjalankan strategi. Berikut adalah kelas dan fungsi yang telah tersedia pada *starter pack*:

Kelas *Bot*

`name: str`
`email: str`
`id: str`

Kelas *Position*
`y: int` `x: int`
Kelas *Properties*

`points: Optional[int] = None`
`pair_id: Optional[str] = None`
`diamonds: Optional[int] = None`
`score: Optional[int] = None`
`name: Optional[str] = None`
`inventory_size: Optional[int] = None`
`can_tackle: Optional[bool] = None`

```
milliseconds_left: Optional[int] = None
time_joined: Optional[str] = None
base: Optional[Base] = None
```

Kelas *GameObject*

```
id: int
position: Position
type: str
properties: Optional[Properties] = None
```

Kelas *Config*

```
generation_ratio: Optional[float] = None
min_ratio_for_generation: Optional[float] = None
red_ratio: Optional[float] = None
seconds: Optional[int] = None
pairs: Optional[int] = None
inventory_size: Optional[int] = None
can_tackle: Optional[bool] = None
```

Kelas *Feature*

```
name: str
config: Optional[Config] = None
```

Kelas *Board*

```
id: int
width: int
height: int
features: List[Feature]
minimum_delay_between_moves: int
game_objects: Optional[List[GameObject]]

# Mengembalikan semua bot yang ada dalam papan permainan
def bots(self) -> List[GameObject]

# Mengembalikan semua diamond yang saat ini ada dalam papan permainan
def diamonds(self) -> List[GameObject]

# Mengembalikan bot pemain
def get_bot(self, bot: Bot) -> Optional[GameObject]

# Mengembalikan true jika move yang diberikan adalah move yang valid
def is_valid_move(self, current_position: Position, delta_x: int,
delta_y: int) -> bool
```

BAB 3
APLIKASI STRATEGI GREEDY

Berdasarkan ketentuan-ketentuan dari permainan yang telah dijelaskan sebelumnya, kami membuat strategi dengan mempertimbangkan situasi-situasi yang mungkin terjadi selama permainan berlangsung agar dapat mengambil *diamond* sebanyak mungkin. Adapun strategi yang kami implementasikan yaitu:

A. *Diamond*

Strategi *diamond* merupakan strategi dengan pendekatan *greedy* yang mempertimbangkan jarak bot ke *diamond* serta poin yang dimiliki *diamond* tersebut (1 poin untuk diamond biru, 2 poin untuk diamond merah). Syarat untuk diaktifkannya strategi ini adalah jika *inventory* bot masih belum penuh, dan waktu bot yang tersisa masih cukup (selengkapnya dijelaskan di strategi *base*). Bila syarat terpenuhi, bot akan menghitung jumlah langkah yang diperlukan untuk menuju masing-masing *diamond* (baik melalui *teleport* maupun tidak), lalu memilih *diamond* yang terdekat. Bila *inventory* bot sudah berisi empat, maka bot juga memperhitungkan jarak *diamond* tersebut ke *base* dengan tujuan memilih diamond yang dekat pula dengan *base*. Serta bila yang terpilih adalah *diamond* biru dan terdapat *diamond* merah yang lebih jauh dua langkah atau kurang, maka bot akan memilih *diamond* merah tersebut.

a. Mapping Elemen Greedy

   1) Himpunan kandidat: Arah gerakan bot ke atas, ke bawah, ke kiri, dan ke kanan

   2) Himpunan solusi: Arah gerakan yang membawa bot menuju *diamond* yang dipilih

   3) Fungsi solusi: Memilih arah gerakan bot supaya bot dapat mencapai *diamond* yang dipilih

   4) Fungsi seleksi:

      - Menghitung jarak bot ke semua kandidat *diamond*, lalu memilih *diamond* yang terdekat dengan bot.

      - Bila isi inventory bot sudah 4, maka jarak yang dipertimbangkan adalah jarak bot menuju diamond ditambah jarak diamond tersebut menuju base.

      - Bila terdapat diamond merah yang selisih jaraknya dengan diamond biru terdekat kurang dari dua langkah, maka bot akan memilih diamond merah tersebut.

5) Fungsi kelayakan: *Diamond* hanya mungkin dipilih jika diamond tersebut adalah diamond biru atau isi *inventory* bot kurang dari sama dengan tiga bila diamond tersebut adalah diamond merah

6) Fungsi objektif: Memaksimalkan diamond yang dapat diambil selama permainan berlangsung

b. Analisis Efisiensi Solusi

Strategi ini menghitung jumlah langkah bot ke semua *diamond* yang ada, baik melewati portal maupun tidak, kemudian memilih *diamond* dengan jarak langkah yang paling sedikit (dengan mempertimbangkan selisih poin diamond dibanding poin diamond sebelumnya). Maka, jumlah komputasi serta perbandingan yang dilakukan program berbanding lurus dengan jumlah diamond yang ada pada waktu tersebut. Sehingga kompleksitas waktu dari strategi ini adalah $O(n)$.

c. Analisis Efektivitas Solusi

Kriteria bot untuk memilih diamond yang ingin dituju adalah jarak bot menuju diamond, jarak bot menuju base, poin yang dimiliki diamond, serta jumlah inventory yang sudah terisi. Bila selisih jarak dari bot antara diamond biru dan diamond merah kurang dari dua langkah, maka bot akan memilih diamond merah yang memiliki poin dua kali lipat diamond biru. Bila inventory bot telah terisi empat, maka bot akan lebih memilih diamond yang jaraknya dekat dari bot serta lebih dekat ke base.

Strategi efektif bila:

- Terdapat diamond biru dan diamond merah yang jaraknya berdekatan
- Saat *inventory* berisi empat, terdapat diamond yang jaraknya dekat dengan base

Strategi tidak efektif bila:

- Diamond yang tidak dipilih oleh bot ternyata memiliki beberapa diamond lain di dekatnya dan selisih jaraknya dengan yang telah dipilih kecil, karena bot tidak mempertimbangkan kerapatan diamond pada suatu area
- Terdapat kumpulan diamond biru yang jaraknya saling berdekatan, namun terdapat diamond merah yang lokasinya berlawanan dengan kumpulan diamond biru tersebut dan selisih jarak diamond merah dibanding jarak diamond biru terdekat kurang dari sama dengan dua langkah.

B. *Red Button*

Strategi *red button* merupakan strategi dengan pendekatan *greedy* yang mempertimbangkan jarak bot ke *red button* serta selisih jarak bot ke *diamond* terdekat dengan jarak bot ke *red button*. Syarat pengaktifan strategi ini adalah *inventory* bot belum penuh dan jarak bot ke *diamond* yang telah dipilih sebelumnya. Bot akan memutuskan untuk menuju *red button* jika jumlah langkah bot ke *red button* empat langkah lebih sedikit dibandingkan jumlah langkah bot ke *diamond* terpilih. Jika *inventory* bot sudah berisi empat, jarak ke *red button* turut memperhitungkan jumlah langkah *red button* ke *base*.

a. Mapping Elemen Greedy

   1) Himpunan kandidat: Arah gerakan bot ke atas, ke bawah, ke kiri, dan ke kanan

   2) Himpunan solusi: Arah gerakan yang membawa bot menuju target *red button*

   3) Fungsi solusi: Memilih arah gerak bot supaya bot dapat mencapai *red button*

   4) Fungsi seleksi: Memilih *red button* hanya jika jumlah langkah menuju *diamond* terdekat lebih banyak empat langkah atau lebih daripada langkah menuju *red button*

   5) Fungsi kelayakan: *Red button* hanya mungkin dipilih jika *inventory* bot belum penuh

   6) Fungsi objektif: Memperbanyak jumlah *diamond* di sekitar bot saat *diamond* terlalu jauh dari bot

b. Analisis Efisiensi Solusi

Strategi ini menghitung jumlah langkah bot ke *red button*, ditambah jarak *red button* ke *base* jika *inventory* berisi 4 *diamond*, kemudian membandingkannya dengan jumlah langkah menuju *diamond* terdekat yang telah dilakukan sebelumnya. Karena hanya ada satu *red button* dalam *map*, maka dilakukan paling banyak empat perhitungan jumlah langkah, yaitu langkah bot menuju *red button*, langkah bot menuju *red button* melalui *teleport*, langkah *red button* menuju *base*, dan langkah *red button* menuju *base* melalui *teleport*. Jadi, strategi ini memiliki kompleksitas konstan atau O(1).

c. Analisis Efektivitas Solusi

Program akan memeriksa jaraknya ke *red button* dan membandingkannya dengan jarak ke *diamond* terdekat, sedangkan ketika *inventory* bot berisi empat, bot akan menghitung serta jarak *red button* ke *base*. Bot akan memilih *red button* jika selisih jarak *red button* yang telah diperhitungkan empat lebih sedikit dibandingkan jarak bot ke *diamond*. Strategi ini berfungsi untuk membawa lebih banyak *diamond* ke sekitar bot.

Strategi efektif bila:

- Di sekitar bot tidak ada *diamond* dan hanya ada *red button*
- Saat *inventory* berisi empat, *red button* berada sangat dekat dengan *base*

Strategi tidak efektif bila:

- *Diamond* di *map* hampir habis dan jarak *diamond* cukup jauh

C. *Teleport*

Strategi *teleport* merupakan strategi dengan pendekatan *greedy* untuk menjadikan *teleport* terdekat sebagai target apabila langkah menuju ke suatu target lebih dekat melalui *teleport*. Strategi ini akan aktif ketika jarak ke *base*, *diamond*, maupun *red button* lebih dekat jika melalui *teleport* dan waktu permainan masih tersisa banyak. Pada awalnya, akan ditentukan *teleport* mana yang terdekat kemudian menghitung langkah menuju *teleport* terdekat dan menghitung jarak target ke *teleport* lainnya. *Teleport* akan dihindari jika target bot saat ini bukanlah *teleport* dan *teleport* menghalangi bot menuju targetnya.

a. Mapping Elemen Greedy

   1) Himpunan kandidat: Arah gerakan bot ke atas, ke bawah, ke kiri, dan ke kanan
   2) Himpunan solusi: Arah gerakan yang membawa bot menuju *teleport* terdekat
   3) Fungsi solusi: Memilih *teleport* terdekat dilanjut dengan memilih arah gerak bot supaya bot mencapai *teleport*
   4) Fungsi seleksi: Waktu masih tersisa banyak dan *teleport* merupakan target saat ini
   5) Fungsi kelayakan: Terdapat target yang lebih dekat jika melalui *teleport*
   6) Fungsi objektif: Memanfaatkan *teleport* untuk memaksimalkan jumlah *diamond* yang dapat diambil dalam waktu yang singkat

b. Analisis Efisiensi Solusi

Strategi ini akan memperhitungkan jarak bot ke *teleport* terdekat lalu menghitung jarak *teleport* lainnya ke seluruh *diamond* dan *red button*. Strategi juga memperhitungkan jarak bot menuju *teleport* terdekat ditambah jarak *teleport* terjauh menuju *base* apabila bot sedang menuju *base* dan waktu masih tersisa banyak. Bila inventory bot sudah terisi empat, maka bot juga akan memperhitungkan jarak target menuju base baik melewati portal maupun tidak. Untuk tiap target yang sedang dihitung jaraknya melalui *teleport*, jumlah perhitungan maksimal yang mungkin diperlukan adalah 4 kali, yaitu jarak bot menuju *teleport* terdekat, jarak *teleport* lainnya menuju target, jarak target menuju base, serta jarak target menuju base melalui *teleport* lainnya. Maka untuk tiap target, kompleksitas waktu yang diperlukan untuk menghitung jarak melalui *teleport* adalah O(1). Bila bot sedang mencari diamond yang ingin dipilih, maka perhitungan jarak

melalui teleport dilakukan sebanyak *n* kali (dengan *n* adalah jumlah diamond yang ada), sehingga waktu totalnya adalah O(*n*).

c. Analisis Efektivitas Solusi

Bot akan mencari terlebih dahulu *teleport* yang paling dekat dengannya, kemudian menghitung jarak *teleport* lainnya ke target yang dipilih sesuai kondisi. Bot memilih *teleport* apabila jarak melalui *teleport* lebih sedikit dibandingkan jarak tanpa melalui *teleport*. Strategi ini berfungsi untuk membantu bot menentukan pilihan terbaiknya. *Teleport* akan dihindari apabila bot tidak memilih *teleport* sebagai targetnya.

Strategi efektif bila:

- Terdapat *diamond-diamond* yang lebih dekat jika melalui *teleport*
- Jarak ke *base* lebih dekat melalui *teleport* ketika bot menuju *teleport*

Strategi tidak efektif bila:

- Bot sedang menuju *teleport*, tetapi *teleport* berpindah tiba-tiba
- Dengan kasus yang sama seperti sebelumnya, dan tidak ada *diamond* di sekitar bot saat ini

D. *Base*

Strategi *base* adalah strategi yang digunakan supaya bot kembali ke *base* untuk mengumpulkan skor. Strategi akan memperhitungkan jarak bot ke *base* lalu bot akan menuju *base*. Syarat pengaktifan strategi ini adalah ketika *inventory* bot sudah penuh (berisi lima *diamond*) atau waktu permainan tersisa sebanyak langkah yang diperlukan bot untuk kembali ke *base*. Jika waktu masih banyak, bot juga akan memperhitungkan jarak ke *base* melalui *teleport*. Sedangkan jika waktu tersisa sedikit, bot tidak akan kembali melalui *teleport*.

a. Mapping Elemen Greedy

1) Himpunan kandidat: Arah gerakan bot ke atas, ke bawah, ke kiri, dan ke kanan

2) Himpunan solusi: Arah gerakan bot yang yang membawa bot kembali ke *base*

3) Fungsi solusi: Mencari arah gerakan bot yang sesuai bergantung pada jumlah langkah antara bot dengan musuh

4) Fungsi seleksi:

- Bila jarak antara musuh dengan bot adalah dua langkah, maka bot akan memilih arah yang bergerak ke musuh untuk mencoba men*tackle*.

- Bila jaraknya tiga langkah, maka bot akan mengganti target *diamond* menjadi *diamond* terdekat yang tidak pada arah yang sama dengan arah musuh.

  Contoh: Bila posisi musuh berada pada kanan atas bot, maka bot akan melangkah menuju *diamond* terdekat yang lokasinya tidak di kanan atas bot.

5) Fungsi kelayakan: Strategi *enemies* hanya mungkin dipilih jika terdapat musuh yang jaraknya dari bot kurang dari sama dengan tiga langkah

6) Fungsi objektif: Bot mengembalikan *diamond* yang sudah dikumpulkannya ke *base* supaya memperoleh skor

b. Analisis Efisiensi Solusi

Bot akan mencari *base*-nya di peta dan menghitung jumlah langkah yang diperlukan bot untuk menuju *base*. Bot pun akan menghitung jumlah langkah menuju *base* melalui *teleport* apabila waktu masih tersisa banyak. Kemudian bot akan membandingkan jumlah langkah tersebut dan memilih yang paling minimal. Bot hanya memiliki satu *base* sehingga untuk setiap tahap hanya akan dilakukan sekali sehingga terdapat tiga kali perhitungan jumlah langkah. Jadi, strategi ini memiliki kompleksitas algoritma yang konstan atau O(1).

c. Analisis Efektivitas Solusi

Bot hanya akan mencari jumlah langkah menuju *base* ketika *inventory* melebihi tiga. Bot otomatis menuju *base* saat *inventory*-nya berisi lima. *Base* tentunya sangat penting supaya *player* dapat memperoleh skor dan menentukan kapan bot harus kembali ke *base* serta bagaimana bot bergerak menuju *base* tentu sangat berpengaruh terhadap jumlah diamond yang berhasil dibawa ke base sebelum permainan berakhir.

Strategi efektif bila:

- *Inventory* bot sudah penuh
- Ada *diamond* yang berada di dekat *base* ketika *inventory* berisi empat.

Strategi tidak efektif bila:

- Bot memiliki kesempatan mengambil *diamond* pada perjalanan pulang, tetapi waktu tersisa sedikit sehingga bot memilih kembali ke *base*
- Waktu tersisa belasan detik dan ada *teleport* di sekitar base dan *teleport* lainnya sangat jauh sehingga bot bisa masuk ke *teleport* tetapi harus kembali karena jaraknya ke base menjauh
- Terdapat *teleport* yang membuat jumlah langkah bot ke *base* lebih singkat

E. *Enemies*

Strategi *enemies* adalah strategi yang digunakan supaya bot mencoba untuk men*tackle* musuh serta menghindari diamond yang ada pada sisi musuh. Strategi akan memperhitungkan jarak bot menuju lawan, lalu jika terdapat lawan yang jaraknya dekat dengan bot maka bot akan merespon sesuai jumlah langkah antara musuh ke bot. Jika jarak dari bot ke musuh adalah 2 langkah, maka bot akan mencoba untuk men*tackle* musuh dengan bergerak ke arah musuh. Sedangkan jika jarak dari bot ke musuh adalah 3 langkah, maka bot akan mengganti target diamond ke diamond yang posisinya tidak searah dengan posisi lawan.

a. Mapping Elemen Greedy

1) Himpunan kandidat: Arah gerakan bot ke atas, ke bawah, ke kiri, dan ke kanan

2) Himpunan solusi: Arah gerakan bot yang yang membawa bot kembali ke *base*

3) Fungsi solusi: Mencari lokasi *base* kemudian menetapkan target bot ke *base*-nya

4) Fungsi seleksi: Memilih jarak terdekat antara menuju base lewat *teleport* atau tidak, atau bila waktu tinggal sedikit maka tidak melewati *teleport*

5) Fungsi kelayakan: Base hanya mungkin dipilih jika *inventory* penuh atau sisa waktu sama dengan jumlah langkah bot ke *base*

6) Fungsi objektif: Bot mencoba untuk men-*tackle* musuh atau mengurangi kemungkinan di-*tackle* oleh bot lain

d. Analisis Efisiensi Solusi

Untuk menjalankan strategi *enemies*, bot akan mengecek satu persatu bot musuh dan menghitung jarak musuh menuju bot. Kemudian bot akan merespon bila terdapat musuh yang jaraknya kurang dari sama dengan tiga langkah dari bot. Bila jaraknya adalah dua langkah, maka bot akan bergerak menuju musuh. Sedangkan bila jaraknya adalah tiga langkah, maka bot akan kembali menghitung jarak menuju semua diamond kemudian memilih diamond berdasarkan strategi diamond diatas yang arahnya tidak sama dengan arah bot menuju musuh. Sehingga maksimal jumlah perhitungan adalah $m+n$ (m adalah jumlah musuh, n adalah jumlah diamond), dengan kompleksitas waktunya adalah $O(m+n)$.

e. Analisis Efektivitas Solusi

Bot hanya akan mengaktifkan strategi *enemies* bila terdapat musuh yang jaraknya kurang dari sama dengan tiga langkah. Kondisi permainan tentunya sangat bervariasi tergantung pada perilaku musuh, sehingga perilaku musuh sangat memengaruhi performa bot kita selama

berjalannya permainan. Namun karena perilaku musuh sulit untuk ditebak (terutama dengan algoritma *greedy* dimana untuk setiap langkah kita tidak dapat melihat kondisi permainan sebelumnya), maka sangat sulit untuk membuat strategi terhadap musuh yang efektif tanpa mengetahui perilaku musuh secara pasti.

Strategi efektif bila:

- Saat musuh berjarak dua langkah, musuh bergerak duluan mengarah ke bot pemain
- Terdapat diamond lain yang jaraknya selisih satu langkah dengan jarak diamond terdekat, dan diamond tersebut tidak searah dengan arah posisi musuh terhadap bot pemain

Strategi tidak efektif bila:

- Saat musuh berjarak dua langkah, musuh tidak bergerak menuju arah bot pemain
- Diamond lain yang tidak searah dengan arah posisi musuh jaraknya cukup jauh dibandingkan diamond yang searah dengan posisi musuh, sehingga setelah bot melangkah menuju diamond tersebut (dan jaraknya menuju musuh tidak lagi tiga langkah) maka bot akan kembali mengejar diamond yang searah dengan posisi musuh walaupun kemungkinan besar musuh akan merebut duluan diamond tersebut

BAB 4
IMPLEMENTASI DAN PENGUJIAN

A. Implementasi dalam *Pseudocode*

Dalam pembuatan bot, dimanfaatkan beberapa kelas, yaitu *Portal*, *Player*, *Diamond*, *Enemies*, *Game State*, *My Bot*, dan beberapa fungsi tambahan. Berikut adalah implementasi algoritma tiap kelas dalam program, disajikan dalam bentuk *pseudocode* agar dapat lebih mudah dipahami.

Kelas *Portal*

```
# Fungsi untuk mencari teleport di map
procedure __init__(input/output self: Teleport,
            input portal_list: List[GameObject], input current_position: Position)
      KAMUS LOKAL
      ALGORITMA
          Hitung jarak antara portal pertama dan portal kedua dengan bot player
          if jarak portal pertama ke bot < jarak portal kedua ke bot then
              self.closest_portal ← portal pertama
              self.Farthest_portal ← portal kedua
          else
               self.closest_portal ← portal kedua
              self.Farthest_portal ← portal pertama

# Fungsi untuk menghitung jarak bot ke teleport
function    count_steps_by_portal(current_position:    Position,    target_position:
Position) → integer
      KAMUS LOKAL
      ALGORITMA
          → jarak bot ke closest_portal + jarak closest_portal ke target

# Fungsi untuk menentukan apakah target lebih dekat jika melalui teleport
function is_closer_by_portal(current_position: Position,
                             target_position: Position) → boolean
      KAMUS LOKAL
          jarak_lewat_portal, jarak_tanpa_portal: integer
      ALGORITMA
          jarak_lewat_portal ← count_steps_by_portal(current_position,
                               target_position)
          jarak_tanpa_portal ← count_steps(current_position, target_position)
          → jarak_lewat_portal < jarak_tanpa_portal
```

Kelas *Player*

```
# Fungsi untuk menginisialisasi atribut player
procedure __init__(output self: Player, input portal_list: List[GameObject], input
```

```
current_position: Position)
      KAMUS LOKAL
      ALGORITMA
          Inisialisasi semua atribut player, dengan atribut boolean bernilai False

# Fungsi untuk menentukan apakah inventory pemain sudah penuh
function is_inventory_full(self: Player) → boolean
      KAMUS LOKAL
      ALGORITMA
          → self.inventory_player = self.inventory_size

# Fungsi untuk mengganti target pemain
procedure set_target(input/output self: Player,input object: GameObject):
      KAMUS LOKAL
      ALGORITMA
          self.current_target ← object
          self.target_position ← object.position

# Fungsi untuk mengganti posisi yang ingin dituju pemain
procedure set_target_position(input/output self, input target_position: Position
      KAMUS LOKAL
      ALGORITMA
          self.target_position ← target_position

# Fungsi untuk mengecek apakah suatu move tidak valid
function is_invalid_move(self: Player, delta_x: integer, delta_y: integer, board:
Board) → boolean
      KAMUS LOKAL
      ALGORITMA
          → (self.current_position.x + delta_x < 0 or self.current_position.x
             delta_x = board.width or self.current_position.y + delta_y < 0 or
             self.current_position.y + delta_y = board.height)

      # Fungsi untuk menghindari rintangan di jalan
procedure avoid_obstacles(input/output self: Player, input portals: Portals, input
is_avoiding_portal: boolean, input board: Board)
      KAMUS LOKAL
          delta_x, delta_y, next_x, next_y: integer
      ALGORITMA
          next_move ← arah menuju target saat ini
          if pada arah tersebut ada teleport and target bot saat ini bukan
             teleport:
              next_move ← arah alternatif menuju target saat ini
              if arah alternatif = arah sebelumnya then
                  next_move ← arah menghindar portal tanpa keluar dari board

          player.next_move ← next_move
```

Kelas *Diamond*

```
procedure __init__(output self: Diamond, output diamonds_list: List[GameObject],
output red_button: GameObject, input diamonds_being_held: int):
```

```
      KAMUS LOKAL
      ALGORITMA
          Menginisialisasi list diamond, red button, dan diamond target

procedure filter_diamond(input/output self: Diamond, input current_position,
           enemy_position: Position)
      KAMUS LOKAL
      ALGORITMA
          if current_position.x ≠ enemy_position.x and
             current_position.y ≠ enemy_position.y then
              Filter diamond pada self.diamonds_list yang tidak searah dengan
              posisi musuh

procedure choose_diamond(input/output self: Diamond, input/output player: Player,
input portals: Portals)
      KAMUS LOKAL
          max_step_diff, diamond_distace, point_diff, i: integer
      ALGORITMA
          max_step_diff ← 2
          i traversal [1..len(self.diamond_list)]
              {Hitung jarak player ke diamond tanpa melalui portal}
              diamond_distance ← count_steps(player.current_position,
              diamond.position)
              if player.diamonds_being_held = 4 then
                  Tambahkan diamond_distance dengan jarak diamond ke base

                  point_diff ← diamond.properties.points -
                               self.chosen_diamond.properties.points
                    if self.chosen_diamond_distance > diamond_distance - (point_diff
*
                       max_step_diff) then
                          Ganti target ke self.diamond_list[i]
          if player di luar portal and step player ke portal < jarak ke diamond
          then
              {Hitung jarak player ke diamond melalui portal}
              i traversal [1..len(self.diamond_list)]
                  diamond_distance ← portals.count_steps_by_portal
                                     (player.current_position, diamond.position)
                  if player.diamonds_being_held = 4 then
                       Tambahkan diamond distance dengan jarak diamond ke base
                      point_diff ← diamond.properties.points -
                                   Self.chosen_diamond.properties.points
                      if self.chosen_diamond_distance > diamond_distance -
                         (point_diff * max_step_diff) then
                          Ganti target ke portal terdekat

procedure  check_red_button(input/output  self:  Diamond,  input/output  player:
Player, input portals: Portal)
      KAMUS LOKAL
          red_button_distance, max_step_diff: integer
      ALGORITMA
          if player di luar portal then
              red_button_distance ← min(jarak player tanpa portal dan dengan
              portal ke red button)
          else
              red_button_distance ← jarak player ke red button tanpa portal
```

```
        if player.diamonds_being_held = 4 then
            Tambahkan jarak red button ke base
        max_step_diff ← 4
        if (red_button_distance ≤ self.chosen_diamond_distance) then
            if portals.is_closer_by_portal(player.current_position,
               self.red_button.position) then
                self.chosen_target ← portals.closest_portal
            else
                self.chosen_target ← self.red_button
```

Kelas *Enemies*

```
# Fungsi untuk menginisialisasi atribut enemies
procedure __init__ (input self, input bots_list: List[GameObject], input
player_bot: GameObject)
    KAMUS LOKAL
    ALGORITMA
       Masukkan bot dari bots_list menuju list_of_enemies kecuali bot player

# Fungsi untuk mengecek apakah ada musuh yang dekat dengan bot
procedure check_nearby_enemy(input/output self, input diamonds: Diamonds, input
player: Player, input portals: Portals, input has_tried_tackle: bool)
    KAMUS LOKAL
        enemy_distance: integer
    ALGORITMA
        i traversal[1..len(list_of_enemies)]
            enemy_distance ← jarak player ke musuh
            if enemy_distance < self.target_enemy_distance then
                self.target_enemy ← list_of_enemies[i]
                self.target_enemy_distance ← enemy_distance

        if self.target_enemy_distance = 2 and not has_tried_tackle and
           musuh berada di posisi diagonal bot player then
            player.next_move ← arah langkah menuju bot musuh
            self.try_tackle ← True

        else if self.target_enemy_distance = 3 then
            self.avoid_enemy(player, diamonds, portals)

# Fungsi untuk mencegah bot bergerak ke arah lawan
procedure avoid_enemy(self, player: Player, diamonds: Diamonds, portals: Portals)
    KAMUS LOKAL
    ALGORITMA
        Eliminasi diamond yang arahnya sama dengan arah bot pemain ke musuh
        Pilih kembali diamond berdasarkan algoritma diamond
        if arah red button tidak sama dengan arah bot pemain ke musuh then
            Cek jarak red button terhadap bot pemain, ambil jika tidak ada
            diamond yang cukup dekat dengan bot pemain
```

Kelas *Game State*

```
procedure __init__(output self: GameState, input board_bot: GameObject, input
board: Board)
    KAMUS LOKAL
    ALGORITMA
        self.board ← board
        self.player_bot ← board_bot

function initialize(self: GameState) → (Player, Diamonds, Portals, Enemies)
    KAMUS LOKAL
        list_of_diamonds: array of Object {Diamond}
        list_of_portals: array of Object {Portal}
        list_of_bots: array of Object {Bot}
        red_button: Object
        i: integer
    ALGORITMA
        list_of_diamonds ← []
        list_of_portals ← []
        list_of_bots ← []
        red_button ← None
        i traversal [1..len(self.board.game_objects)]
            if self.board.game_objects[i] adalah Diamond then
                Masukkan ke list_of_diamonds
            else if self.board.game_objects[i] adalah Portal then
                Masukkan ke list_of_portals
            else if self.board.game_objects[i] adalah red_button then
                red_button ← self.board.game_objects[i]
            else
                Masukkan ke list_of_bots
        → (Player(self.player_bot.position, self.player_bot.properties),
           Diamonds(list_of_diamonds, red_button,
                    self.player_bot.properties.diamonds),
           Portals(list_of_portals, self.player_bot.position),
           Enemies(list_of_bots, self.player_bot.position))

function no_time_left(self: GameState, current_position, base_position: Position) →
boolean
        → (not position_equals(current_position, base_position) and
            self.player_bot.properties.milliseconds_left /
            count_steps(current_position, base_position) ≤ 1300)
```

Kelas *My Bot*

```
procedure __init__(output self: MyBot)
    KAMUS LOKAL
    ALGORITMA
        self.back_to_base ← false
        self.is_avoiding_portal ← false
        self.tackle ← false

function next_move(self: MyBot, board_bot: Object, board: Board) → (integer,
```

```
integer)
     KAMUS LOKAL
          game_state: GameState
          player: Player
          diamonds: Diamond
          portals: Portal
          enemies: Enemy
     ALGORITMA
          game_state ← GameState(board_bot, board)
          player, diamonds, portals, enemies = game_state.initialize()

          if position_equals(player.current_position, player.base_position) then
              self.back_to_base ← false
          else if position_equals(player.current_position,
                  portals.closest_portal.position) then
              player.is_inside_portal ← true

          if self.back_to_base or player.is_inventory_full() or
             game_state.no_time_left(player.current_position,player.base_position)
          then
              player.set_target_position(player.base_position)
              self.back_to_base ← true

              if (not player.is_inside_portal and
              (not game_state.no_time_left(player.current_position,
               player.base_position) or count_steps(player.current_positon,
               portals.closest_portal.positon) = 1) and
               portals.is_closer_by_portal(player.current_position,
               player.base_positon)) then
                  player.set_target(portals.closest_portal)
          else
              diamonds.choose_diamond(player, portals)
              if diamonds.chosen_target then
                  diamonds.check_red_button(player, portals)
                  player.set_target(diamonds.chosen_target)
              else
                  player.set_target_position(player.base_position)
                  player.back_to_base ← true
          if not game_state(no_time_left(player.current_position,
           player.base_position)) then
              enemies.check_nearby_enemy(diamonds, player, portals, self.tackle)
              self.tackle ← enemies.try_tackle
          if not enemies.try_tackle then
              player.avoid_obstacle(portals, self.is_avoiding_portal, board)
              self.is_avoiding_portal ← player.is_avoiding_portal
          → player.next_move
```

Fungsi Tambahan

```
function get_direction_alt(current_x, current_y, dest_x, dest_y: integer) →
          (integer, integer)
     KAMUS LOKAL
```

```
        delta_x, delta_y: integer
    ALGORITMA
        delta_x ← clamp(dest_x - current_x, -1, 1)
        delta_y ← clamp(dest_y - current_y, -1, 1)
        if delta_y != 0 then
            delta_x ← 0
        → delta_x, delta_y

function count_steps(a: Position, b: Position) → integer
    KAMUS LOKAL
    ALGORITMA
        → abs(a.x - b.x) + abs(a.y - b.y)

function coordinat_equals(x1, x2, y1, y2: integer) → boolean
    KAMUS LOKAL
    ALGORITMA
        → x1 = x2 and y1 = y2

function same_direction(pivot, a, b: Position) → boolean
    KAMUS LOKAL
    ALGORITMA
        if a.x ≥ pivot.x and a.y ≥ pivot.y then
            → b.x ≥ pivot.x and b.y ≥ pivot.y
        else if a.x ≥ pivot.x and a.y ≤ pivot.y then
            → b.x ≥ pivot.x and b.y ≤ pivot.y
        else if a.x ≤ pivot.x and a.y ≥ pivot.y then
            → b.x ≤ pivot.x and b.y ≥ pivot.y
        else if a.x ≤ pivot.x and a.y ≤ pivot.y then
            → b.x ≤ pivot.x and b.y ≤ pivot.y
```

B. Struktur Data Program

Untuk merealisasikan program dan cara kerja bot, kami membuat beberapa kelas baru untuk mempermudah dan merapikan proses pembuatan program. Berikut adalah kelas-kelas baru yang kami buat:

Kelas *Portal*

```
closest_portal: GameObject
farthest_portal: GameObject

# Fungsi untuk mencari teleport di map
def __init__(self, portal_list: List[GameObject], current_position:
Position)

# Fungsi untuk menghitung jarak bot ke teleport
def count_steps_by_portal(self, current_position: Position,
target_position: Position) -> int

# Fungsi untuk menentukan apakah target lebih dekat jika lewatteleport
def is_closer_by_portal(self, current_position: Position,
```

```
    target_position: Position) -> bool
```

Kelas *Portal* mencakup fungsi untuk mengimplementasi strategi Teleport. *Portal* terdiri atas atribut “closest_portal” dan “farthest_portal”. Closest_portal dan farthest_portal menyimpan portal yang lebih dekat dan lebih jauh dengan bot. *Portal* terdiri atas 2 method, yaitu “count_steps_by_portal” dan “is_closer_by_portal”. Count_steps_by_portal berfungsi mengembalikan jarak antara bot dan portal. Is_closer_by_portal berfungsi membandingkan jarak menuju target dengan melalui portal dan dengan tidak melalui portal. Is_closer_by_portal mengembalikan True apabila jarak menuju target lebih dekat apabila melalui portal dan False apabila jarak menuju target lebih jauh apabila melalui portal.

Kelas *Player*

```
current_position: Position
target_position: Optional[Position]
base_position: Position
next_move: Tuple[int, int]
current_target: Optional[GameObject]
inventory_size: int
diamonds_being_held: int
is_inside_portal: bool
is_avoiding_portal: bool
milliseconds_left: int

# Fungsi untuk menginisialisasi properti pemain
def __init__(self, current_position: Position, props: Properties):
Position)

# Fungsi untuk menentukan apakah inventory pemain sudah penuh
def is_inventory_full(self) -> bool

# Fungsi untuk mengganti target pemain
def set_target(self, object: GameObject):

# Fungsi untuk mengganti posisi yang ingin dituju pemain
def set_target_position(self, target_position: Position)

# Fungsi untuk mengecek apakah suatu move tidak valid
def is_invalid_move(self, delta_x: int, delta_y: int, board: Board)

# Fungsi untuk menghindari rintangan di jalan
def avoid_obstacles(self, portals: Portals, is_avoiding_portal: bool,
board: Board)
```

Kelas *Player* mencakup fungsi-fungsi utilitas bagi bot. Player terdiri atas atribut “current_position”, “target_position”, “base_position”, “next_move”, “current_target”, “inventory_size”, “diamonds_being_held”, “is_inside_portal”, “is_avoiding_portal”, dan “milliseconds_left”. Current_position berfungsi menyimpan posisi bot sekarang, target_position berfungsi menyimpan posisi target, base_position berfungsi menyimpan posisi base, next_move

berfungsi menyimpan gerakan selanjutnya bot (x,y), current_target berfungsi menyimpan komponen yang menjadi target bot, inventory_size berfungsi menyimpan kapasitas inventory, diamonds_being_held berfungsi menyimpan jumlah diamond yang berada di inventory, is_inside_portal berfungsi menandakan apakah bot berada di dalam teleporter, is_avoiding_portal berfungsi menandakan apakah bot sedang berusaha menghindari rintangan berupa portal, dan milliseconds_left berfungsi menyimpan waktu yang tersisa pada round. Player terdiri atas method "is_inventory_full", "set_target", "set_target_position", "is_invalid_move", dan "avoid_obstacles". Is_inventory_full berfungsi untuk menentukan apakah inventory pemain sudah penuh. Set_target berfungsi mengganti target pemain dengan mengubah atribut current_target dan set_target_position berfungsi mengubah atribut target_position. Is_invalid_move berfungsi memeriksa apakah suatu move tidak valid. Avoid_obstacles berfungsi mengatur next_move untuk menghindari suatu rintangan/komponen di jalan.

Kelas *Diamond*

```
diamonds_list: List[GameObject]
chosen_diamond: GameObject
chosen_diamond_distance: int
chosen_target: GameObject
red_button: GameObject

# Fungsi untuk mencari diamond dan red button di map
def __init__(self, diamonds_list: List[GameObject], red_button:
GameObject, diamonds_being_held: int)

# Fungsi untuk memilih diamond yang terdekat dengan bot
def choose_diamond(self, player: Player, portals: Portals)

# Fungsi untuk menentukan pengambilan red button
def check_red_button(self, player: Player, portals: Portals)
```

Kelas *Diamond* mencakup fungsi untuk mengimplementasi strategi Diamond. Diamond terdiri atas atribut "diamonds_list", "chosen_diamond", "chosen_diamond_distance", "chosen_target", dan "red_button". Diamond_list berfungsi menyimpan semua diamond di dalam board yang valid (diamond merah tidak dimasukkan ke dalam list apabila bot membawa empat diamond), chosen_diamond berfungsi menyimpan diamond yang akan dijadikan target bot, chosen_diamond_distance berfungsi menyimpan jarak bot terhadap chosen_diamond, chosen_target berfungsi menyimpan komponen yang menjadi target akhir bot (antara portal, diamond, atau red button), dan red_button berfungsi menyimpan lokasi red button. Diamond terdiri atas method "filter_diamond", "choose_diamond", dan "check_red_button". Filter_diamond berfungsi untuk memilih diamond yang tidak berada di arah kita menuju musuh. Choose_diamond berfungsi memilih diamond yang menjadi target sementara sesuai dengan strategi untuk komponen diamond. Check_red_button berfungsi memeriksa apakah lebih efektif untuk mengambil red button sesuai dengan strategi untuk komponen red button.

Kelas *Enemies*

```
enemies_list: List[GameObject]
target_enemy: GameObject
target_enemy_distance: int
enemy_target_position: Position
try_tackle: bool
wait_move: bool


# Fungsi untuk menginisialisasi atribut enemies
def __init__(self, bots_list: List[GameObject], player_bot: GameObject)

# Fungsi untuk mengecek apakah ada musuh yang dekat dengan bot
def check_nearby_enemy(self, diamonds: Diamonds, player: Player,
portals: Portals, has_tried_tackle: bool)

# Fungsi untuk mencegah bot bergerak ke arah lawan
def avoid_enemy(self, player: Player, diamonds: Diamonds, portals:
Portals)
```

Kelas *Enemies* mencakup fungsi untuk mengimplementasi strategi Enemies. Enemies terdiri atas atribut "enemies_list", "target_enemy", "target_enemy_distance", "enemy_target_position", dan "try_tackle". Try_tackle berfungsi untuk mengubah arah gerak bot menuju bot musuh bila jarak dari bot ke musuh adalah dua langkah. Kelas *enemies* terdiri dari tiga method, yaitu "check_nearby_enemy" dan "avoid_enemy". Method check_nearby_enemy digunakan untuk memeriksa apakah terdapat musuh yang dekat dengan bot, dan mengarahkan bot jika jaraknya sama dengan dua langkah. Fungsi "avoid_enemy" dijalankan jika terdapat musuh yang jaraknya tiga langkah dari bot, dimana program akan kembali memilih diamond yang arahnya tidak sama dengan arah musuh terhadap pemain.

Kelas *GameState*

```
board: Board
player_bot: GameObject

# Fungsi untuk membaca map dan bot
def __init__(self, board_bot: GameObject, board: Board)

# Fungsi untuk menginisiasi seluruh kondisi permainan
def initialize(self)

# Fungsi untuk menghitung ketika bot harus kembali ke base
def no_time_left(self, current_position: Position, base_position:
Position) → bool
```

Kelas *GameState* mencakup fungsi untuk menginisialisasi algoritma bot setiap kali bot mendapat giliran, serta untuk menangani hal-hal terkait *state* permainan saat bot mendapat giliran. Kelas *GameState* terdiri dari dua atribut, yaitu *board* dan *player_bot*. Atribut *board*

digunakan untuk menyimpan state board pada giliran pemain, dan *player_bot* digunakan untuk menyimpan bot pemain sebagai objek. *GameState* memiliki tiga method, yaitu *init*, *initialize*, dan *no_time_left*. Method *init* digunakan untuk menginisialisasi atribut *GameState*, *initialize* digunakan untuk menginisialisasi kelas lain yang telah diimplementasikan (kelas Portals, Player, Diamonds, Enemies), dan fungsi *no_time_left* akan mengembalikan *true* jika waktu yang tersisa sudah tersisa sedikit untuk bot pemain kembali ke *base*.

Kelas *MyBot*

```
# Fungsi untuk menginisialisasi bot
def __init__(self)

# Fungsi untuk mencari target selanjutnya
 def next_move(self, board_bot: GameObject, board: Board) -> Tuple[int,
 int]
```

Kelas *MyBot* adalah kelas utama pada permainan, dan merupakan kelas yang menghubungkan semua kelas lainnya. MyBot hanya terdiri dari inisialisasi dan method "next_move". Next_move berfungsi menghubungkan method-method yang telah dirancang pada semua kelas lainnya. MyBot berfungsi mengatur semua kelas untuk membangun logika bot secara keseluruhan.

C. Pengujian

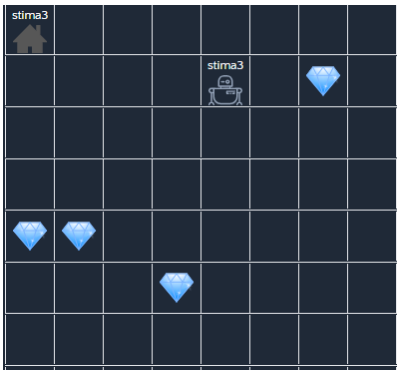
Setelah mengimplementasikan algoritma strategi yang telah dibuat, dilakukan pengujian keberjalanan strategi dengan menjalankan bot pada permainan bersama dengan beberapa bot lain. Selama permainan berlangsung, perilaku bot pemain diamati serta dievaluasi apakah tindakan yang dilakukan bot sesuai dengan strategi yang telah kami rancang sebelumnya. Berikut merupakan hasil dari pengujian strategi bot pada permainan:

1) Strategi mencari diamond terdekat

   Pada gambar dibawah, bot mengkalkulasi jarak bot menuju masing-masing diamond. Kemudian bot akan memilih untuk bergerak menuju diamond yang jaraknya lebih dekat, yaitu diamond yang posisinya dibawah.

Gambar 4.1.1 Ilustrasi Strategi Pencarian Diamond Terdekat

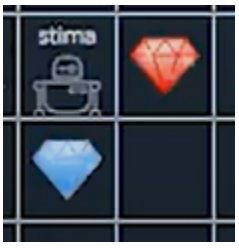
Namun, terdapat kekurangan yaitu bot tidak dapat mengkalkulasi area di mana terdapat kumpulan diamond atau kepadatan diamond tinggi. Akibatnya, bot tetap akan menuju diamond terdekat dan kumpulan diamond dapat diambil oleh lawan.



Gambar 4.1.2 Bot Tidak Mengejar Kumpulan Diamond

2) Strategi prioritas *red diamond*

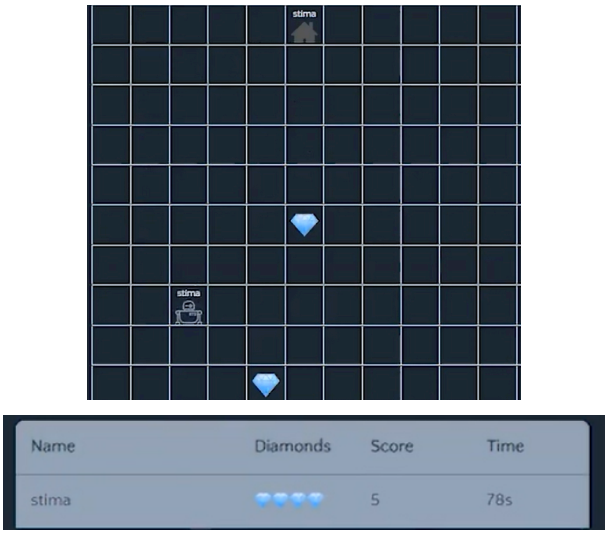
Setelah mengkalkulasi jarak dan menentukan diamond terdekat, bot akan membandingkan diamond terdekat tersebut dengan red diamond terdekat. Bila diamond yang paling dekat adalah diamond biru dan selisih antara jarak menuju diamond biru dengan jarak menuju diamond merah kurang dari sama dengan dua langkah, maka bot akan memilih untuk menuju diamond merah.



Gambar 4.2 Ilustrasi Strategi Prioritas Red Diamond

3) Strategi mengambil diamond terdekat dengan *base*

Ketika *inventory* bot sudah terisi empat, maka bot juga akan memperhitungkan jarak diamond menuju base (sehingga jarak yang dibandingkan adalah jarak bot menuju diamond ditambah jarak diamond menuju base) agar bot mengambil diamond yang dekat dengan bot serta letaknya berdekatan dengan base. Hal ini bertujuan untuk meminimalisir kemungkinan bot di-tackle oleh musuh dalam perjalanan pulang, serta memperoleh lebih banyak diamond bila musuh menekan red button ketika pemain sudah dekat dengan base, sehingga pemain dapat mengosongkan inventory dengan cepat lalu mengejar diamond yang baru muncul.



Gambar 4.3 Ilustrasi Strategi Prioritas Diamond yang dekat dengan Base

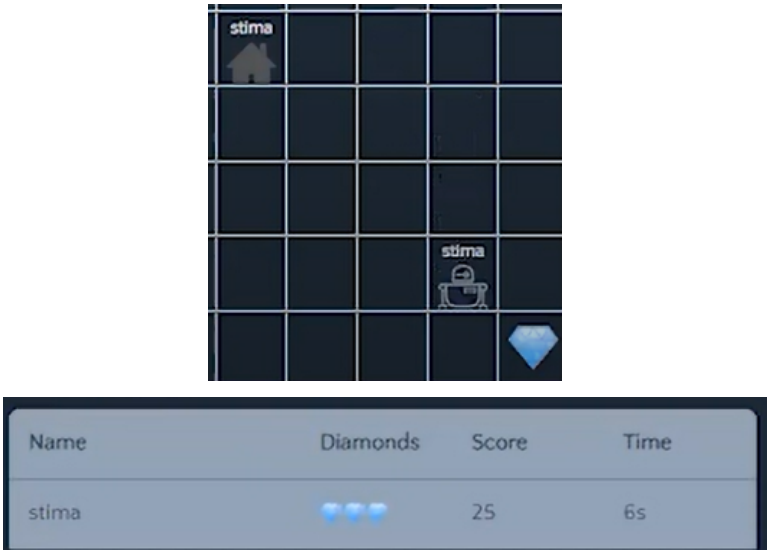
4) Strategi kembali ke base saat inventory penuh

Setelah inventory bot penuh, bot akan langsung menuju ke base. Bot tidak akan menuju diamond walaupun jaraknya dekat.

Gambar 4.4 Ilustrasi Strategi Kembali ke Base Saat Inventory Penuh

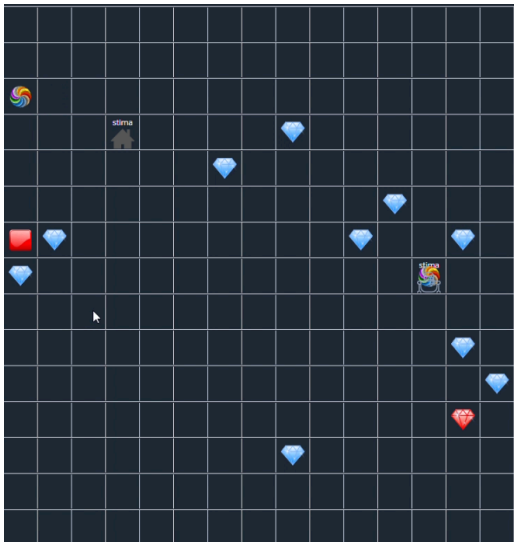
5) Strategi waktu hampir habis

Ketika waktu hampir habis, bot akan mengkalkulasi waktu yang dibutuhkan untuk menuju ke base. Apabila waktu sudah tersisa sedikit bagi bot untuk kembali ke base (dihitung dari rasio waktu permainan yang tersisa dibagi jarak bot menuju base), maka bot akan berhenti mengambil diamond dan bergerak menuju base.



Gambar 4.5.1 Ilustrasi Strategi Kembali ke Base Saat Waktu Hampir Habis

Bot akan kembali ke base ketika waktu tersisa sedikit dengan memperhitungkan jarak dari bot menuju base tanpa melewati portal. Sehingga pada kasus dimana bot melewati portal untuk mengejar diamond, bila jarak portal tersebut terlalu jauh dari base maka setelah bot masuk portal dan dekat dengan diamond, bot akan masuk portal lagi untuk kembali menuju base tanpa mengambil diamond walaupun waktu yang tersisa sebenarnya masih cukup untuk mengambil diamond jika yang diperhitungkan adalah jarak dari bot menuju base melewati portal.



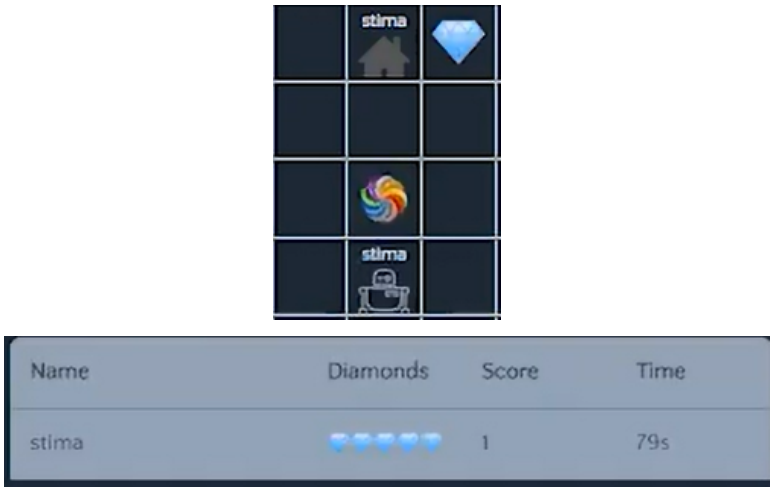
Gambar 4.5.2 Kondisi Bot Masuk-Keluar Portal Tanpa Mengambil Diamond

Pada gambar di bawah, untuk menuju base, bot akan bergerak ke kanan lalu ke atas, bukan bergerak ke atas lalu ke kanan lalu ke atas lagi. Padahal, bot sebenarnya dapat bergerak ke atas dan mengambil diamond sambil bergerak menuju base. Pada kalkulasi strategi waktu hampir habis, bot belum bisa mengkalkulasi rute supaya bot dapat mengambil diamond sambil bergerak menuju base.

Gambar 4.5.3 Kembali ke Base Tanpa Mengambil Diamond

6) Strategi menghindari *teleport*

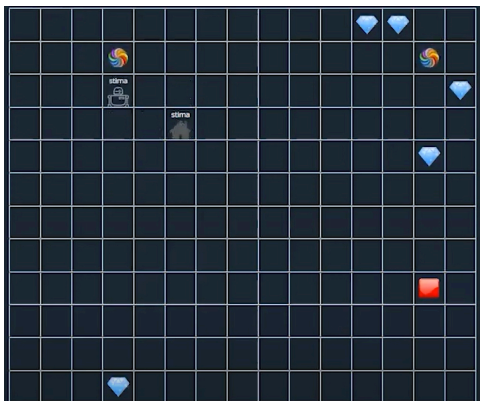
Pada gambar di bawah, bot memiliki target berupa diamond. Apabila terdapat rintangan teleporter di jalur menuju diamond, bot akan menghindari teleporter sehingga jarak bot menuju diamond tidak semakin jauh.



Gambar 4.5 Ilustrasi Strategi Menghindar Teleport Jika Target Bukan Teleport

7) Strategi Teleport

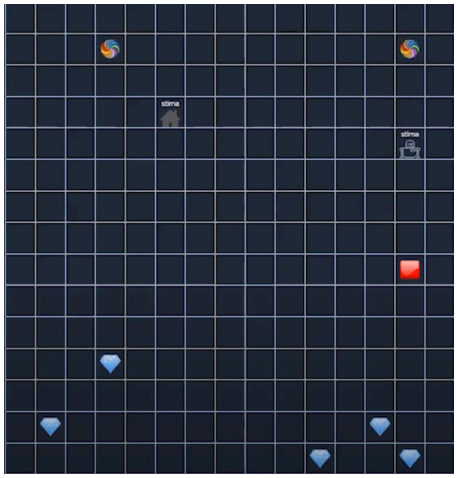
Bot akan mengkalkulasi jarak diamond di sekitar bot dan di sekitar teleporter. Apabila jarak diamond di sekitar teleporter lebih dekat, bot akan memilih untuk mengambil teleporter.



Gambar 4.7 Ilustrasi Strategi Teleport

8) Strategi Red Button

Apabila jarak red button dengan diamond lain sangat jauh, bot akan mencoba mengambil red button dengan harapan setelah board ter-*reset*, akan terdapat diamond yang lebih dekat dengan bot.



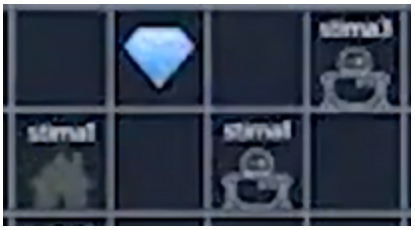
Gambar 4.8.1 Ilustrasi Strategi Red Button

Pada kasus dibawah, terdapat diamond yang letaknya lebih dekat dengan red button, namun pada area diamond tersebut sudah ada bot lawan yang lebih dekat dengan diamond tersebut dan sedang merebut diamond. Pada kondisi ini akan lebih menguntungkan bila bot pemain menekan red button, namun karena masih terdapat diamond yang relatif dekat dengan bot maka bot akan memilih untuk mengejar diamond tersebut.

Gambar 4.8.2 Red Button yang Tidak Efektif

9) Strategi Tackle

Pada kasus dibawah, jarak bot pemain dengan bot musuh adalah dua langkah. Bot akan memilih untuk mengarah ke bot musuh dengan harapan agar dapat men-*tackle* musuh. Namun, strategi ini tidak selalu efektif karena tidak ada jaminan bahwa bot musuh akan bergerak ke arah bot pemain.



Gambar 4.9 Ilustrasi Strategi Tackle

10) Strategi Menghindari Bot Lawan

Sedangkan pada kasus di bawah, jarak bot pemain dengan bot musuh adalah tiga langkah. Pada awalnya, bot akan lebih memprioritaskan untuk mengambil red diamond. Namun karena terdapat musuh dengan jarak tiga langkah, maka bot akan mencari diamond lain yang dekat dengan bot namun tidak searah dengan posisi musuh terhadap bot. Dalam kasus ini, bot akan lebih memilih untuk bergerak ke diamond biru yang letaknya tepat di kiri bot pemain. Namun, strategi ini tidak terlalu efektif bila tidak terdapat diamond yang cukup dekat selain diamond yang searah dengan musuh, karena pada giliran selanjutnya bila jarak musuh sudah lebih dari tiga langkah maka bot akan kembali mengejar diamond yang searah dengan posisi musuh.

Gambar 4.10 Ilustrasi Strategi Menghindari Bot Lawan

BAB 5

KESIMPULAN DAN SARAN

A. Kesimpulan

Walaupun telah menerapkan strategi *greedy*, bot yang telah dibuat masih memiliki kelebihan dan kekurangan pada area-area tertentu. Bot cukup cepat dalam mengambil keputusan sehingga juga cepat melakukan gerakan. Bot juga pintar dalam memanfaatkan teleporter untuk menuju lokasi target. Bot juga pintar dalam menggunakan red button ketika tidak ada diamond di sekitar bot. Bot juga dapat mengkalkulasi waktu yang dibutuhkan untuk kembali ke base sehingga bot mengetahui kapan harus kembali ke base ketika sisa waktu sudah sedikit. Bot juga dapat mempertimpangkan musuh yang berada di dekatnya. Apabila bot berjarak 2 langkah dari musuh, bot akan mencoba untuk men-*tackle* musuh. Sedangkan ketika bot berjarak 3 langkah dari musuh, bot akan menjauh jika *diamond* yang diincarnya searah dengan arah musuh. Di sisi lain, bot tidak efektif dalam memprediksi lokasi dengan kepadatan diamond tertinggi. Bot juga tidak memprediksi apakah red button akan dilangkahi sehingga bot dapat memboroskan langkahnya ketika mencoba menuju komponen yang akan hilang. Bot juga tidak memprediksi apakah musuh akan melewati teleporter sehingga bot mungkin di-*tackle* oleh musuh atau diamond yang diincar bot dapat dicuri oleh musuh. Bot juga tidak sempurna dalam mengkalkulasi waktu yang dibutuhkan untuk menuju ke base sehingga apabila ada diamond yang dapat diambil dalam perjalanan pulang, bot mungkin tidak mengambil diamond tersebut.

B. Saran

Saran untuk bot yang kami buat adalah sebagai berikut:

1. Pengembangan algoritma dilakukan dengan mempertimbangkan segala kemungkinan sehingga menambah keoptimalan strategi *greedy*
2. Penambahan komentar pada *source code* program supaya program lebih mudah dibaca oleh teman sekelompok maupun orang lain
3. Perencanaan pengembangan program dapat ditingkatkan supaya proses pembuatan bot lebih efektif

LAMPIRAN

Tautan repository:

https://github.com/BryanLauw/Tubes1_tbfoReborn.git

Tautan video:

https://youtu.be/sImZPJ8HHx0

DAFTAR PUSTAKA

Munir, R. (n.d.). Homepage Rinaldi Munir. https://informatika.stei.itb.ac.id/~rinaldi.munir/