LAPORAN TUGAS BESAR I
STRATEGI ALGORITMA

Kelompok Nama Kelompoknya Nanti:
M. Athaullah Daffa Kusuma M (13522044)
Rafiki Prawhira Harianto (13522065)
Julian Caleb Simandjuntak (13522099)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

Diamonds adalah sebuah programming challenge berbasis game di mana pemain akan melakukan pemrograman pada sebuah bot (yang merepresentasikan player) untuk mengumpulkan diamond sebanyak-banyaknya, dengan beberapa *gimmick* dan rintangan yang seru. Program permainan Diamond dibagi menjadi 2, yaitu game engine yang terdiri atas backend dan frontend, dan bot starter pack yang terdiri atas pemanggilan API, bot logic, dan program utama. Pengerjaan Tugas Besar Strategi Algoritma 1 ini dilakukan dengan cara mahasiswa, sebagai pemain, akan membuat bagian program bot logic dengan strategi greedy, mengembangkan starter pack yang sudah ada pada bot.

Untuk game Diamonds sendiri, terdapat beberapa komponen meliputi diamonds yang dikumpulkan, red/diamond button yang mengacak posisi diamond, teleporters, bots dan bases untuk menyetor diamond, dan inventory. Objektif utama bot adalah mengambil diamond sebanyak mungkin (dengan 2 poin untuk diamond merah dan 1 poin untuk diamond biru) dan menyetornya di base, tetapi bot dapat menimpa (tackle) bot lain untuk mengambil diamond yang dimiliki bot lain. Permainan dilakukan hingga waktu seluruh bot telah berakhir.

BAB II
LANDASAN TEORI

Dasar Teori

Algoritma Greedy (bahasa Indonesia: rakus) merupakan sebuah algoritma yang memecahkan persoalan secara langkah (step by step) sedemikian sehingga pada setiap langkahnya, program dapat mengambil pilihan yang terbaik yang dapat diperoleh saat itu tanpa memikirkan konsekuensi kedepan, dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma greedy digunakan untuk memecahkan persoalan optimasi, yaitu mencari nilai ekstrem, baik maksimasi maupun minimasi.

Algoritma greedy mempunyai 6 elemen, yaitu

- Himpunan kandidat (C): kandidat yang akan dipilih dari setiap langkah
- Himpunan solusi (S): kandidat yang dipilih
- Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
- Fungsi seleksi: memilih kandidat berdasarkan strategi greedy tertentu yang bersifat heuristik
- Fungsi kelayakan: memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi atau tidak
- Fungsi obyektif: memaksimumkan atau meminimumkan

Algoritma greedy melibatkan pencarian sebuah himpunan bagian S dari himpunan kandidat C di mana S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan dioptimisasi oleh fungsi obyektif.

Cara Kerja Program

Program dibagi menjadi 2, yaitu program untuk board dan program untuk bot.

Board

Program untuk board akan dijelaskan secara singkat. Board yang digunakan dalam permainan Diamonds berukuran 15x15 satuan persegi, dibuat dengan menggunakan Node.js dan Docker untuk databasenya. Selama permainan, board tersebut akan diisi dengan beragam komponen yang berupa bot para players dengan masing-masing

basenya, sejumlah diamond biru dan merah yang diletakkan secara random, diamond button yang memberikan diamond lebih banyak, dan sepasang teleporter.
Komponen-komponen pada board (dan bot) diimplementasikan dengan pendekatan Object-oriented dengan atribut (properti) masing-masing yang nantinya dapat diakses pada program untuk bot.

```typescript
export interface IBotGameObjectDto {
  type: "BotGameObject" | "DummyBotGameObject";
  position: Position;
  properties: BotGameObjectProperties;
}

export interface IDiamondGameObjectDto {
  type: "DiamondGameObject";
  position: Position;
  properties: DiamondGameObjectProperties;
}

export interface IDiamondButtonGameObjectDto {
  type: "DiamondButtonGameObject";
  position: Position;
  properties: {};
}

export interface IBaseGameObjectDto {
  type: "BaseGameObject";
  position: Position;
  properties: BaseGameObjectProperties;
}

export interface ITeleportGameObjectDto {
  type: "TeleportGameObject";
  position: Position;
  properties: TeleportProperties;
}

export type IGameObjectDto =
  | IBotGameObjectDto
  | IDiamondGameObjectDto
  | IDiamondButtonGameObjectDto
  | IBaseGameObjectDto
  | ITeleportGameObjectDto;
```

```typescript
export type BotGameObjectProperties = {
  base: Position;
  diamonds: number;
  timeJoined: Date;
  inventorySize: number;
  canTackle: boolean;
  score: number;
  name: string;
  nextMoveAvailableAt: Date;
  millisecondsLeft: number;
};
```

Bot

Program untuk bot dibuat dengan menggunakan Python yang memiliki API (api.py) untuk mendapatkan segala informasi tentang board dari program board. API juga akan digunakan untuk mengirim informasi yang sudah diproses pada program bot kembali kepada board untuk diolah. Template repository yang diberikan telah menyediakan code untuk mengambil bot dan diamond yang dijadikan list.

```python
@property
def bots(self) -> List[GameObject]:
    return [d for d in self.game_objects if d.type == "BotGameObject"]

@property
def diamonds(self) -> List[GameObject]:
    return [d for d in self.game_objects if d.type == "DiamondGameObject"]
```

Algoritma bot diletakkan pada folder logic, dengan nama bot menjadi nama file berekstensi .py. Untuk membuat bot baru, buat file .py baru dan mengimplementasikan algoritma (termasuk algoritma greedy yang menjadi target tubes ini) dalam paradigma OOP. Kemudian, bot tersebut didaftarkan pada CONTROLLERS yang terdapat pada file main.py.

```
init()
BASE_URL = "http://localhost:3000/api"
DEFAULT_BOARD_ID = 1
CONTROLLERS = {
    "Random": RandomLogic
}
```

Bot mempunyai beberapa atribut, seperti point yang dimiliki, jumlah diamond yang dimiliki (belum disetor ke base), dan waktu hidup. Bot dapat bergerak ke 4 arah (dx, dy) yaitu WEST (-1, 0), EAST (1, 0), NORTH (0, 1), dan SOUTH (0, -1); dx dan dy dalam 1 satuan persegi. Selain ke-4 move tersebut, dan untuk move yang tidak mungkin seperti out of bounds, dicek dan dipastikan invalid oleh program.

Bot juga mempunyai atribut goal_position, yang menerima komponen lain, seperti base dan diamond. Menggunakan atribut ini, program menghitung jarak antara bot dan goalnya dengan menggunakan fungsi get_direction, kemudian dilakukan update setiap tick untuk memindahkan bot tersebut berdasarkan valid move yang telah didefinisikan sebelumnya.

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

Bot dijalankan dengan syntax,

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

yaitu dengan menerima parameter bot yang dijalankan, email, nama, password, dan team. Email dan password tidak harus valid.

BAB III
APLIKASI STRATEGI GREEDY

Mapping Persoalan

Pada tugas besar ini, akan dibuat bot yang mampu mengambil diamond sebanyak-banyaknya pada permainan Diamonds selama waktu tertentu. Jika dilakukan mapping persoalan dalam elemen-elemen algoritma greedy:

- Himpunan kandidat: Himpunan lintasan dari bot ke diamond-diamond (dan komponen lain) yang terdapat pada board
- Himpunan solusi: Himpunan lintasan yang terpilih berdasarkan fungsi solusi
- Fungsi solusi: Memeriksa apakah jumlah diamond yang dapat diambil bot adalah maksimal sebelum waktu habis
- Fungsi seleksi: Memilih lintasan yang terpendek antara bot dan diamond, dengan memastikan bot masih bisa mengambil diamond
- Fungsi kelayakan: Memeriksa apakah bot masih bisa mengambil diamond dan apakah gerakan yang dilakukan oleh bot valid
- Fungsi obyektif: Panjang lintasan minimum untuk jumlah diamond maksimum

Eksplorasi Solusi Greedy

Dikarenakan persoalan permainan Diamonds ini melibatkan banyak komponen, dapat dibuat banyak solusi greedy yang dapat diimplementasikan.

1. Mengambil semua diamond terdekat

Untuk setiap langkah, dari himpunan kandidat akan diambil lintasan antara bot dan diamond yang terdekat/terkecil, kemudian bot akan bergerak menuju diamond tersebut. Langkah ini akan diulangi sebanyak 5 kali, yaitu jumlah maksimal diamond yang bisa dibawa oleh bot. Bot lalu ke base untuk menyetor diamond. Setelah disetor, bot mengulangi langkah pengambilan diamond. Langkah-langkah diatas diulangi hingga waktu bot habis. Himpunan solusi akan berisi lintasan-lintasan antara bot dan diamond yang terpendek.

2. Variasi solusi 1 dengan pivot sejumlah diamond

Solusi ini merupakan alternatif solusi 1, di mana setelah mengambil sejumlah diamond tertentu, misal 3, bot akan kembali ke base, jika jarak bot ke base lebih

dekat daripada jarak bot ke diamond-diamond. Hal ini dilakukan untuk mencegah kemungkinan dicuri oleh bot lain dan waktu habis sebelum bot menyetor diamond.

3. Variasi solusi 1 dengan mengambil diamond yang dekat ketika pulang ke base

Solusi ini merupakan solusi sebelumnya dengan tambahan algoritma untuk bot menyempatkan mengambil diamond apabila selama perjalanannya ke base menemukan diamond yang dekat, semisal berjarak 2 satuan persegi dari dirinya.

4. Mengambil diamond terdekat dengan pertimbangan teleporter

Solusi 4 adalah alternatif lain solusi 1 tetapi dengan mempertimbangkan lintasan bot ke diamond melalui teleporter. Misal untuk pasangan teleporter A dan B, apabila ada sebuah lintasan antara teleporter A dengan diamond ditambah antara teleporter B ke bot (A tidak sama dengan B) yang lebih pendek daripada jarak bot ke diamond lainnya, lintasan tersebut akan diambil.

5. Mengambil diamond terdekat dengan pertimbangan bot lain

Solusi 5 adalah alternatif lain solusi 1 tetapi dengan mempertimbangkan jika ada kesempatan bagi bot untuk mencuri diamond dari bot lain. Pencurian tersebut akan dilakukan apabila bot lebih dekat dengan bot musuh dibandingkan dengan diamond ataupun base. Juga akan dibuat sebuah batasan, misalnya pencurian diamond hanya dilakukan bila bot musuh mempunyai diamond lebih banyak daripada diamond yang dimiliki bot sendiri.

6. Mengambil diamond terdekat dengan pertimbangan teleporter dan bot lain

Solusi 6 adalah gabungan dari solusi 4 dan 5, yaitu mempertimbangkan jarak lintasan antara bot ke diamond di sekitar, antara bot ke diamond yang lintasannya melalui teleporter, dan antara bot dengan bot musuh, kemudian diambil lintasan mana yang paling pendek.

7. Mengambil diamond terdekat namun dengan mencari lintasan penuh terlebih dahulu

Solusi ini adalah solusi yang menciptakan graf dengan bot dan diamonds (maupun komponen lainnya) sebagai vertex atau simpul dan lintasannya sebagai edges atau sisi. Lintasan dibuat dari base atau bot berada, ke diamond-diamond di board, kembali ke base. Solusi ini memakan waktu lebih lama karena memikirkan semua kemungkinan lintasan antar komponen dan belum tentu paling optimal.

Analisis Efisiensi dan Efektivitas

Analisis efisiensi dan efektivitas setiap alternatif solusi adalah sebagai berikut

Solusi	O(n)	Penjelasan
1	O(n)	Solusi ini adalah solusi paling sederhana, dengan n adalah semua posisi diamond selain bot, kemudian dihitung jaraknya ke bot, dan dicari yang terpendek. Solusi ini akan menjadi basis untuk analisis solusi berikutnya.
2	O(n)	Solusi ini lebih cepat daripada solusi 1, karena hanya melakukan sejumlah langkah (hingga pivot), lalu hanya perlu mencari lintasan ke base, tidak perlu membandingkan lagi. Solusi ini mempunyai kompleksitas algoritma yang sama dengan solusi 1.
3	O(n)	Solusi ini lebih lambat dari solusi 2, masih mempunyai kompleksitas algoritma yang sama, hanya dengan perhitungan tambahan setiap langkah.
4	O(n)	Solusi ini lebih kompleks daripada solusi 1, di mana akan melakukan perhitungan tambahan, yaitu antara kedua teleporter dengan bot maupun dengan diamond. Kompleksitas algoritma tetap sama.
5	O(n)	Solusi ini juga lebih kompleks daripada solusi 1, di mana akan melakukan perhitungan tambahan, yaitu antara bot sendiri dengan bot musuh. Solusi ini juga mempunyai kompleksitas algoritma yang sama.
6	O(n)	Solusi ini adalah gabungan dari solusi 4 dan 5, yang pastinya lebih kompleks. Namun, masih mempunyai kompleksitas algoritma yang sama, karena tidak terdapat loop di dalam loop.
7	O(n^2)	Solusi ini adalah solusi yang paling kompleks dikarenakan adanya perbandingan antar komponen, tidak hanya bot, kemudian mencari lintasan-lintasan yang terdekat membentuk lintasan gabungan.

Kompleksitas algoritma yang dihasilkan solusi 1 - 6 adalah O(n), bukan O(n^2), karena hanya akan dilakukan perhitungan jarak dan perbandingan lintasan antara bot dengan masing-masing komponen, bukan antar seluruh komponen.
Kompleksitas algoritma yang dihasilkan solusi 7 adalah O(n^2), di mana dilakukan perbandingan antar komponen yang terdapat pada board, lalu dicari total kemungkinan seluruh lintasan yang terpendek.

Strategi Greedy yang Dipilih

Berdasarkan analisis efisiensi, analisis efektivitas, waktu dan sumber daya yang dimiliki selama tubes, kelompok kami memilih untuk menerapkan strategi algoritma greedy dengan ketentuan sebagai berikut:

1. Kami akan menggunakan alternatif solusi 5, yaitu solusi mengambil diamond-diamond terdekat dari bot dengan mempertimbangan lintasan yang memanfaatkan teleporter.
2. Pada solusi ini, kami akan menggunakan pivot diamond, yaitu 3 diamond, untuk mencegah kehilangan diamond oleh bot maupun waktu.
3. Untuk lebih memaksimalkan diamond, kami juga akan memampukan bot untuk mengambil diamond yang jaraknya dekat dengan bot selama melalui lintasan pulang ke base (jika inventory belum penuh), seperti pada solusi 3.
4. Bot juga akan memperhitungkan dan mengambil apabila ada diamond merah, yang mana rewardnya lebih banyak, berada di sekitarnya, meskipun sedikit lebih jauh daripada diamond biru.
5. Terakhir, akan dibuat sedemikian sehingga bot pulang ke base dan menyetor diamond sebelum bot tersebut mati.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

Pseudocode Algoritma Greedy:

```
function next_move(bot, papan) -> position
{Mengembalikan gerakan bot berikutnya berdasarkan algoritma greedy}

Deklarasi:
  goalPos : posisi akhir

Algoritma:
if (jumlah diamond < 3) and (waktu untuk pulang cukup):
  findNearestDiamond()
  goalPos = {jarak terdekat ke diamond melalui teleporterA, teleporterB, atau
langsung ke diamond}

else: {bot pulang}
  findNearestDiamondAlt()
  if (saat pulang, terdapat diamond dengan jarak <= 2) and (waktu untuk pulang
cukup):
    goalPos = {jarak terdekat ke diamond melalui teleporterA, teleporterB,
atau langsung ke diamond}
  else:
    goalPos = {jarak terdekat ke base melalui teleporterA, teleporterB, atau
langsung ke base}

validasi(goalPos)
-> goalPos
```

```
function findNearestDiamond()
{Menghasilkan posisi diamond terdekat berdasarkan jaraknya ke bot, atau
diamond merah terdekat jika selisih jaraknya dengan diamond terdekat <=2}
```

```
function findNearestDiamondAlt()
{Menghasilkan posisi diamond terdekat berdasarkan jaraknya ke bot saja}
```

Penjelasan:

- listDiamond merupakan sebuah larik yang berisi object dari *diamond-diamond* yang ada di papan permainan, variabel ini berbentuk *Array of GameObject*
- listTeleporters merupakan sebuah larik yang berisi object dari *teleporters-teleporters* yang ada di papan permainan, variabel ini berbentuk *Array of GameObject*
- current_position merupakan posisi dari bot sekarang, variabel ini berbentuk *Position*

- goalPos merupakan posisi akhir yang hendak dituju oleh bot, variabel ini berbentuk *Position*
- *GameObject* merupakan tipe data yang mendeklarasikan semua objek yang ada di papan permainan ini. Objek tersebut dapat melambangkan diamond, teleporters, bot, base, atau komponen apa saja yang ada di papan permainan ini. Game*Object* memiliki atribut id, *Position*, tipenya, dan *Properties*
- *Position* merupakan tipe data yang menjelaskan posisi dari suatu informasi. *Position* mempunyai atribut berupa posisi x (int) dan posisi y (int).
- *Properties* merupakan tipe data terkecil yang menyimpan informasi yang dimiliki objek-objek yang ada di papan permainan, seperti points yang menyatakan points yang telah didapat bot, diamonds menyatakan berapa diamond yang dibawa bot, score menyatakan berapa score yang dimiliki diamond tersebut, dan banyak atribut lainnya yang menjelaskan apa yang dimiliki objek tersebut.

Analisis Desain Solusi

Solusi algoritma greedy ini cukup optimal ketika kita hendak menghindari lawan, dan ingin mengambil diamond sebanyaknya dengan aman. Solusi greedy ini optimal dan aman dari pencuri untuk kasus dimana diamond berkumpul dan bot kita dapat mengambil cukup banyak, lalu pulang, dan saat perjalanan pulang, bot kita dapat mengambil lagi hingga inventory penuh dan pulang dengan aman. Akan tetapi, solusi ini mempunyai kelemahan, diantaranya yang telah kita uji dan lihat, solusi ini tidak optimal ketika:

- Saat perjalanan pulang, kita dapat menemui teleporter di jalan dan menginjaknya. Dengan begitu kita perlu langkah keluar dan kembali ke teleporter untuk menghindari error dimana delta_x dan delta_y bernilai 0. Hal ini tentu akan menghabiskan waktu sebanyak 2-3 langkah yang terbuang.
- Saat kita menggunakan teleporter untuk mengambil diamond, teleporter dapat berubah lokasinya secara acak. Di sini kita akan pulang melalui jalan lain karena teleporter sebelumnya hilang, membuat bot kita akan mengambil langkah yang cukup jauh dan tidak efisien.
- Saat kasus diamond jauh dan hanya diamond button yang dekat, di sini diamond button akan kita ignore karena diamond button dikira tidak masuk

strategi greedy kita dan akan menyebabkan beberapa corner case. Mungkin akan lebih optimal jika bot menginjak button terlebih dahulu baru mencari diamond lagi.

- Saat kita hendak diserang bot lain. Di sini greedy kita tidak memperhitungkan bot lain, mengakibatkan bot kita rawan di serang oleh bot lain.

BAB V
KESIMPULAN DAN SARAN

Pada akhirnya, algoritma *greedy* tidak selalu solusi yang optimal. *Greedy* hanya mengambil jalan yang terlihat paling optimal pada saat ini, dengan harapan jalan tersebut akan diakhiri oleh solusi yang optimal, atau setidaknya suboptimal. Walaupun begitu, kompleksitas waktu dari *greedy* mungkin jauh lebih cepat dibanding algoritma yang mendapatkan solusi yang optimal, seperti bruteforce, beberapa kasus divide and conquer, dan algoritma lainnya.

Bot yang kami rancang dengan algoritma greedy yang kami buat juga bukanlah solusi optimal untuk mendapatkan *diamond* yang banyak dalam waktu yang terbatas. Walaupun bot kami mengambil *diamond* yang banyak selama dalam jarak tertentu dan mengembalikannya ke rumah dengan konsiderasi *teleporter*, Bot kami akan tidak optimal ketika bertemu bot lainnya, dan ketika diamond terlalu jauh dan button reset dekat (Bot kami akan mengabaikannya).

Saran kita untuk pelajar algoritma greedy adalah mencoba berbagai pendekatan greedy dalam menyelesaikan masalah, karena berfikir heuristik merupakan salah satu dasar dalam menemukan strategi greedy yang baik.

LAMPIRAN

- Repository Github: https://github.com/Julian-Caleb/Tubes1_Nama-Kelompoknya-Nanti
- Video Tubes 1 Stima - Kelompok "Nama Kelompoknya Nanti": https://youtu.be/J-N5puHQymY

DAFTAR PUSTAKA

- Munir, Rinaldi. 2024. "Algoritma Greedy (Bagian 1)" https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf
- Munir, Rinaldi. 2024. "Algoritma Greedy (Bagian 2)" https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf
- Munir, Rinaldi. 2024. "Algoritma Greedy (Bagian 3)" https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf