LAPORAN TUGAS BESAR
IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma *Greedy* dalam Permainan Etimo Diamond

Disusun oleh:

Yusuf Ardian Sandi	13522015
Moh Fairuz Alauddin Yahya	13522057
Rayhan Fadhlan Azka	13522095

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

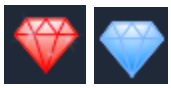
BAB I
DESKRIPSI TUGAS

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

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

Komponen-komponen dari permainan Diamonds antara lain:

1. Diamond



Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio antara diamond merah dan biru ini akan berubah setiap regeneration.

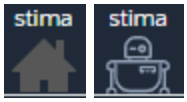
2. Red Button/Diamond Button

Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

3. Teleporters

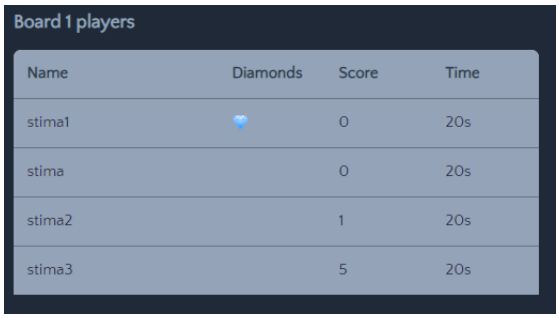
Terdapat 2 teleporter yang saling terhubung satu sama lain. Jika bot melewati sebuah teleporter maka bot akan berpindah menuju posisi teleporter yang lain.

4. Bots and Bases



Pada game ini kita akan menggerakkan bot untuk mendapatkan diamond sebanyak banyaknya. Semua bot memiliki sebuah Base dimana Base ini akan digunakan untuk menyimpan diamond yang sedang dibawa. Apabila diamond disimpan ke base, score bot akan bertambah senilai diamond yang dibawa dan inventory (akan dijelaskan di bawah) bot menjadi kosong.

5. Inventory



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

Untuk mengetahui flow dari game ini, berikut ini adalah cara kerja permainan Diamonds.

1. Pertama, setiap pemain (bot) akan ditempatkan pada board secara random. Masing-masing bot akan mempunyai home base, serta memiliki score dan inventory awal bernilai nol.
2. Setiap bot diberikan waktu untuk bergerak, waktu yang diberikan semua sama untuk setiap pemain.
3. Objektif utama bot adalah mengambil diamond-diamond yang ada di peta sebanyak-banyaknya. Seperti yang sudah disebutkan di atas, diamond yang berwarna merah memiliki 2 poin dan diamond yang berwarna biru memiliki 1 poin.
4. Setiap bot juga memiliki sebuah inventory, dimana inventory berfungsi sebagai tempat penyimpanan sementara diamond yang telah diambil. Inventory ini sewaktu-waktu bisa penuh, maka dari itu bot harus segera kembali ke home base.
5. Apabila bot menuju ke posisi home base, score bot akan bertambah senilai diamond yang tersimpan pada inventory dan inventory bot akan menjadi kosong kembali.
6. Usahakan agar bot anda tidak bertemu dengan bot lawan. Jika bot A menimpa posisi bot B, bot B akan dikirim ke home base dan semua diamond pada inventory bot B akan hilang, diambil masuk ke inventory bot A (istilahnya tackle).
7. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila anda menuju posisi objek tersebut.
8. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

BAB II
LANDASAN TEORI

1. Algoritma Greedy

Algoritma Greedy adalah algoritma yang sering digunakan untuk optimasi. Algoritma ini memilih pilihan yang dianggap paling optimal pada saat tertentu, tanpa mempertimbangkan konsekuensi di masa depan. Berbeda dengan pendekatan konservatif yang memperhatikan efek jangka panjang, algoritma greedy lebih fokus pada keuntungan segera.

Biasanya, algoritma greedy diterapkan dengan pendekatan heuristik, yang menentukan parameter apa yang harus diperhatikan dalam memilih solusi yang optimal. Meskipun tidak selalu menjamin solusi yang mutlak optimal, algoritma greedy dapat menghasilkan solusi lokal yang mendekati solusi global jika diberi waktu yang cukup.

2. Cara Kerja Program

Mengacu pada spesifikasi, eksekusi aksi bot pada kelompok kami berada pada 1 file logic saya , dimana bot akan memilih aksi beserta hasil arah aksi tersebut berdasarkan *priority* yang telah ditetapkan, dalam kasus ini adalah algoritma greedy.

Perubahan yang dilakukan oleh kelompok kami adalah bagaimana cara menentukan aksi selanjutnya dengan mengimplementasikan algoritma greedy ke dalamnya melalui *method* bernama *next_move* dalam kelas *AlucardGreedy*. Di dalam metode ini, proses yang diperlukan oleh algoritma greedy untuk memilih langkah yang paling optimal pada kondisi permainan dilakukan. Kami telah menetapkan bahwa algoritma yang kami gunakan adalah greedy by *priority*, yang mana akan memilih aksi terbaik sebagai langkah yang paling optimal pada saat itu berdasarkan prioritasnya, dan kemudian melaksanakannya.

Ada batasan yang kami terapkan pada algoritma greedy kami untuk memastikan bahwa ia tetap memenuhi definisi algoritma greedy. Salah satunya adalah batasan pada penggunaan waktu, di mana bot kami akan selalu mengevaluasi kondisi permainan saat ini dan tidak memperhitungkan hasil langkah yang telah dilakukan pada kondisi permainan sebelumnya. Selain itu, bot juga tidak memiliki kemampuan untuk memprediksi kondisi permainan di masa depan.

3 komponen penting dalam memahami cara kerja game ini, yaitu :

1. Engine, komponen yang berperan dalam mengimplementasikan logic dan rules game.
2. Runner, komponen yang berperan dalam pelaksana sebuah match serta menghubungkan bot dengan engine.
3. Bot, komponen yang berperan sebagai peserta dalam suatu match

Garis besar cara kerja program game Etimo lokal adalah sebagai berikut:

1. Runner–saat dijalankan– akan meng-host sebuah match pada sebuah hostname tertentu. Untuk koneksi lokal, runner akan meng-host pada localhost:8082 dengan mengambil database lokal. Runner akan menampilkan papan game state
2. Engine kemudian dijalankan untuk melakukan koneksi dengan runner melalui API gateway.

3. Bot akan mengirimakan request melalai API pada runner dan mengubah dari setiap pergerakan bot, setiap API url punya kegunaannya masing-masing.
4. Game akan berjalan hingga waktu berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar.

Berdasarkan gambaran cara kerja program game yang telah disebutkan sebelumnya, berikut merupakan cara menjalankan game secara lokal:

A. Game Engine
   1. Persyaratan Instalasi Node.JS, Docker Desktop,Yarn
   2. Instalasi dan Konfigurasi Awal:
      - Download source code (.zip) pada release game engine.
      - Extract zip dan masuk ke folder hasil extract.
      - Buka terminal dan masuk ke root directory project.
      - Install dependencies dengan Yarn: yarn
   3. Setup environment variable default:
      - Untuk Windows: ./scripts/copy-env.bat
      - Untuk Linux / macOS: chmod +x ./scripts/copy-env.sh dan ./scripts/copy-env.sh
   4. Setup local database:
      - Buka Docker Desktop.
      - Jalankan di terminal: docker compose up -d database
      - Jalankan script: Untuk Windows: “./scripts/setup-db-prisma.bat” dan untuk Linux / macOS: “chmod +x ./scripts/setup-db-prisma.sh dan ./scripts/setup-db-prisma.sh”
   5. Build dengan perintah “npm run build”
   6. Run dengan perintah “npm run start”

B. Bot

1. Download source code (.zip) pada release bot starter pack dan extract zip dan masuk ke folder hasil extract.
2. Install dependencies: pip install -r requirements.txt
3. Lakukan perintah untuk menjalankan program bot:
   - Menjalankan satu bot: python main.py --logic Random --email=your_email@example.com --name=your_name --password=your_password --team etimo
   - Menjalankan beberapa bot:
     - Untuk Windows: ./run-bots.bat
     - Untuk Linux / macOS: ./run-bots.sh

BAB III
APLIKASI STRATEGI GREEDY

1. Mapping Persoalan Algoritma Greedy dalam Permainan Diamonds

Permainan diamonds dapat didekomposisi secara umum menjadi beberapa strategi, yaitu : Mencari diamond terbanyak, Mencari tujuan paling optimal (diamond,base, teleport,red button), dan Mencari rute terdekat. Terdapat beberapa pendekatan dalam menerapkan ketiga strategi tersebut, kami melakukan dekomposisi masalah menjadi beberapa greedy sebagai berikut :

A. Greedy by Nearest Diamond

Pada strategi greedy ini kami melakukan seleksi terhadap objek objek yang ada di board lalu mengumpulkan seluruh list yang berisi posisi-posisi diamond baik itu blue diamond maupun red diamond yang memiliki posisi yang berbeda-beda. Dari seluruh diamond tersebut, akan diambil diamond dengan jarak terpendek dari posisi bot kami. Adapun elemen-elemen yang diperlukan dalam algoritma greedy by nearest diamond adalah

1. Himpunan kandidat : Objek red diamond dan blue diamond
2. Himpunan solusi : objek red diamond atau blue diamond dengan jarak terpendek
3. Fungsi solusi : mengembalikan objek diamond dengan jarak terpendek
4. Fungsi seleksi : Melakukan pengecekan pada setiap red diamond atau blue diamond
5. Fungsi kelayakan : memeriksa apakah diamond dengan poin tersebut dapat diambil (bisa masuk ke inventori)
6. Fungsi obyektif : Diamond merupakan diamond terdekat

B. Greedy by Best Cluster

Pada strategi greedy ini kami melakukan seleksi terhadap wilayah pada board yang memiliki diamond dengan jumlah terbanyak. Kami membagi board menjadi 4 wilayah dengan ukuran wilayah yang sama, lalu bot mencari wilayah dengan diamond terbanyak dan bot akan pergi ke diamond terdekat pada wilayah tersebut. Elemen yang diperlukan dalam algoritma greedy by best cluster ini adalah :

1. Himpunan kandidat : objek red diamond dan blue diamond
2. Himpunan solusi : objek red diamond atau blue diamond yang terletak pada cluster dengan diamond terbanyak
3. Fungsi solusi : mengembalikan objek diamond pada cluster dengan diamond terbanyak dan merupakan diamond terdekat pada cluster tersebut
4. Fungsi seleksi : mengelompokkan setiap diamond berdasarkan cluster, lalu mencari cluster dengan diamond terbanyak lalu mencari diamond terdekat pada cluster tersebut
5. Fungsi kelayakan : memeriksa apakah diamond dengan poin tersebut dapat diambil (bisa masuk ke inventori)

6. Fungsi obyektif : diamond merupakan diamond yang terletak pada cluster dengan diamond terbanyak dan merupakan diamond terdekat pada cluster tersebut

C. Greedy by Nearest Diamond Relative to Base

Pada algoritma ini kami mengimplementasikan penggabungan dari greedy by nearest diamond, greedy by best cluster, dan optimasi tertentu. Pada greedy ini, kami melakukan looping terhadap seluruh diamond, lalu anggaplah diamond yang sedang diproses sebagai diamond_1, diamond yang terdekat dengan diamond_1 dengan diamond_2, poin yang diperoleh jika mengambil diamond_1 sebagai diamond_1_point, poin yang diperoleh jika mengambil diamond_2 sebagai diamond_2_point dan posisi bot sebagai bot_position dan posisi base sebagai base . Lalu bot akan mengcalculate distance dengan formula :

$$Distance = jarak(bot\_position, diamond\_1) + jarak(diamond\_1, base) +$$

$$jarak(diamond\_1, diamond\_2) - diamond\_1\_point - diamond\_2\_point$$

Rumus distance tidak hanya mencari diamond terdekat, tetapi juga mempertimbangkan jarak diamond tersebut ke base, hal ini dilakukan karena jika mempertimbangkan jarak diamond terdekat tanpa mempertimbangkan jarak diamond tersebut ke base, bot mungkin akan berjalan lebih jauh saat ingin pulang ke base untuk setor diamond sehingga menghabiskan lebih banyak waktu. Selain itu, rumus distance ini juga mempertimbangkan jarak diamond tersebut ke diamond terdekat lain terhadap diamond tersebut, hal ini dilakukan agar bot akan mencari diamond yang memiliki diamond lain didekatnya, hal ini merupakan simplifikasi dari greedy by cluster yang dimana greedy by cluster mengecek suatu area dan jumlah diamondnya, tapi disini hanya mengecek dua buah diamond saja. Selain itu terdapat pengurangan terhadap poin diamond 1 dan diamond 2, hal ini bertujuan untuk lebih memprioritaskan diamond yang memiliki poin yang lebih tinggi.

Setelah melakukan looping terhadap seluruh diamond dan mendapat distancenya, bot akan mengeset target posisinya menuju diamond dengan distance terpendek. Beberapa elemen yang diperlukan pada greedy by nearest diamond relative to base ini adalah :

1. Himpunan kandidat : objek red diamond dan blue diamond
2. Himpunan solusi : posisi objek diamond yang paling optimal, sesuai penjelasan diatas
3. Fungsi solusi : mengembalikan posisi objek diamond dengan pertimbangan jarak, jarak ke base, poin, dan jarak ke diamond lain
4. Fungsi seleksi : mempertimbangkan distance masing-masing diamond
5. Fungsi kelayakan : memeriksa apakah diamond dengan poin tersebut dapat diambil (bisa masuk ke inventori)
6. Fungsi obyektif : Posisi diamond yang didapat optimal

D. Greedy to Find Shortest Route

Greedy ini sederhana, kami hanya mempertimbangkan rute terpendek jika bot berjalan seperti biasa dan bot berjalan sambil menggunakan portal. Beberapa elemen yang diperlukan pada greedy to find shortest route ini adalah :

1. Himpunan kandidat : objek portal, dan posisi target awal
2. Himpunan solusi : posisi target akhir dengan rute optimal (sama seperti posisi target awal, atau posisi portal terdekat)
3. Fungsi solusi : mengembalikan posisi menuju ke portal terdekat, atau langsung ke posisi target
4. Fungsi seleksi : memilih rute dengan jarak terpendek dengan cara memilih minimum distance dari (bot ke target pos) dan (bot ke near portal + far portal ke target pos)
5. Fungsi layak : memeriksa apakah rute tersebut lebih pendek
6. Fungsi obyektif : rute yang dipilih terpendek

2. Eksplorasi Alternatif Solusi

A. Machine Learning

Salah satu solusi alternatif untuk meningkatkan kinerja bot adalah dengan menerapkan ***machine learning*** (ML). Dengan ML, bot dapat mengeksplorasi berbagai kemungkinan dalam mengambil *diamond* secara optimal, baik dari segi jumlah maupun kecepatan. ML juga dapat memilih pendekatan yang paling efektif dan efisien, dengan menguji berbagai strategi pada awalnya dan memilih yang paling sukses. Pendekatan ini dapat dikategorikan sebagai *greedy* karena bertujuan untuk memilih pendekatan yang menghasilkan *diamond* terbanyak dan tercepat. Namun, dalam proses eksplorasi awal, mungkin diperlukan *backtracking* untuk mengevaluasi dan meninggalkan strategi yang tidak efektif atau efisien. Jika ML menemui strategi yang tidak memuaskan pada suatu percobaan, maka akan melakukan *backtracking* untuk mencoba pendekatan yang berbeda.

B. Greedy by Enemy Bot

Solusi alternatif lain yang kami tawarkan adalah penambahan ***greedy by enemy bot***. Solusi ini selain berfokus mengambil *diamond* sebanyak-banyaknya dan secepat-cepatnya, juga berfokus untuk men-*tackle* bot musuh dan menghindari musuh yang berusaha men-*tackle*. Tidak seperti solusi kami yang hanya berharap pada keberuntungan untuk men-*tackle* musuh dan menghindar dari *tackle* musuh, justru solusi ini memaksimalkan fitur *canTackle* yang diadakan pada permainan ini. Dalam solusi ini, terdapat dua metode, pertama saat terdapat musuh yang telah mengumpulkan lima poin *diamond* pada *inventory*-nya, maka kita akan segera menuju *base* musuh tersebut (atau sekitarnya). Selanjutnya, ketika musuh telah berada di depan muka, kita langsung segera bergerak ke posisi musuh tersebut untuk men-*tackle*-nya. Metode kedua, saat kita sibuk mengumpulkan *diamond*, tetapi ada musuh yang mendekati kita dalam jarak satu kotak. Dengan begitu, kita akan berhenti sejenak dan setelah musuh telah berada di depan muka, kita langsung segera bergerak ke posisi musuh tersebut untuk men-*tackle*-nya. Walaupun demikian, solusi ini belum pasti dapat men-*tackle* musuh dengan tepat waktu karena sejauh ini

kami tidak dapat menemukan cara untuk memperkirakan siapa yang akan bergerak lebih dulu (bot musuh atau kita). Oleh karena itu, terdapat kemungkinan justru bot musuh-lah yang akan men-*tackle* kita.

Strategi ***greedy by enemy bot*** masih dapat diimplementasikan dengan metode lain yang lebih non-agresif. Strategi ini memperbolehkan kita untuk tidak mengejar diamond yang sia-sia. Diamond yang sia-sia ini didefinisikan sebagai diamond yang dikejar oleh musuh dan berada paling dekat dengan kita jika dibandingkan dengan diamond lain. Namun di sisi lain, bot musuh lebih dekat dengan diamond tersebut sehingga sangat mungkin bahwa musuh tersebut bisa meraih diamond itu terlebih dahulu. Oleh karena itu, di saat yang sama lebih baik bot menargetkan diamond lain agar perjalanan yang ditempuh tidak sia-sia.

C. Greedy by Weight

Strategi ***greedy by weight*** juga dapat menjadi pilihan solusi alternatif untuk memenangkan permainan Diamonds ini. Dalam hal ini, weight didefinisikan sebagai banyaknya poin yang didapat setiap langkah yang diambil sehingga setiap diamond akan memiliki weight berdasarkan perhitungan

$$weight = \frac{diamond's\ point}{number\ of\ steps}$$

Setiap diamond akan dihitung bobotnya dan pada akhirnya akan dipilih diamond dengan bobot terbesar. Namun, strategi ini memiliki kekurangan, yaitu terdapat kemungkinan dimana diamond-diamond yang memiliki bobot besar semua terletak jauh dari base sehingga membutuhkan waktu yang lebih lama untuk mengumpulkan poin-poin tersebut. Selain harus kembali ke base yang jaraknya jauh, saat itu juga terdapat kemungkinan di-*tackle* oleh musuh lain.

D. Greedy by Best Cluster

Selain ***greedy by weight***, ***greedy by best cluster*** juga merupakan salah satu alternatif yang dapat dijadikan solusi. Algoritma greedy ini berjalan dengan cara membagi wilayah pada board menjadi beberapa bagian (dalam hal ini kami telah mencoba membagi menjadi 4 bagian atau 4 kuadran), setelah membagi menjadi beberapa kuadran, akan dihitung kuadran mana dengan jumlah diamond terbanyak sekaligus mencari diamond terdekat dari bot kami di masing-masing kuadran. Setelah didapat kuadran dengan jumlah diamond terbanyak maka bot kami akan pergi ke diamond terdekat yang ada di kuadran tersebut. Strategi ini jadinya tidak kami pilih karena jika mencari diamond dengan kuadran terbanyak, letak kuadrannya dapat terletak terlalu jauh dari base dan lokasi kita, sehingga hal ini tidak efektif karena bot akan berjalan terlalu jauh dan terdapat resiko yaitu diamondnya bisa saja dimakan terlebih dahulu oleh bot musuh.

E. Solusi Terpilih

Terakhir, solusi alternatif yang kami sediakan merupakan ***greedy by distance ,game condition priority***, ***and diamond's point***. Solusi ini nantinya yang akan kami gunakan dan telah dijelaskan pada bagian "Strategi Greedy yang Dipilih"

3. Analisis Efisiensi dan Efektifitas

A. Analisis terhadap Greedy by Enemy Bot

Strategi ini mengutamakan mentackle musuh untuk mendapatkan diamond. Pada percobaan yang telah kami lakukan dengan menggunakan strategi ini, kami mendapatkan bahwa ketika kami berusaha untuk mentackle musuh, kami kesulitan untuk mengejar mereka karena pergerakan dalam permainan terdapat delay yang sudah di set, sehingga tackle yang dilakukan tidak konsisten. Sama halnya ketika kita menghindari musuh agar bot kita tidak tertackle, hal ini juga tidak konsisten karena bisa saja musuh mentackle kita karena bot kita tidak sempat untuk bergerak kabur. Selain itu, algoritma ini juga kurang efektif karena pada algoritma ini, kami mendapat bahwa bot kami malah lebih sering menghindar daripada mencari diamond, sehingga meminimalkan diamond yang didapat.

B. Analisis terhadap Greedy by Weight

Strategi ini mengutamakan bot untuk mengambil diamond berdasarkan bobotnya, yaitu poin diamond dibagi dengan jarak. Strategi ini menurut kami kurang efektif karena tidak mempertimbangkan jarak dari diamond ke base sehingga seringkali bot *roaming* terlalu jauh dan menyebabkan bot membutuhkan banyak waktu ketika ingin setor diamond ke base.

C. Analisis terhadap Greedy by Best Cluster

Pada strategi greedy by best cluster, bot akan pergi menuju cluster dengan diamond terbanyak, strategi ini terkadang kurang efektif karena cluster dengan diamond terbanyak bisa saja jaraknya jauh dari base, ataupun cluster tersebut sudah terdapat bot lain yang berada disana, jadi ketika bot kami sudah sampai cluster tersebut, diamond disana kebanyakan sudah diambil oleh bot lain.

4. Strategi Greedy yang Dipilih

Dalam rangka mencapai tujuan permainan ini, yaitu mengumpulkan sebanyak mungkin diamond dan menyimpannya dalam inventory, solusi optimal menurut kami adalah menggunakan kombinasi algoritma ***greedy by distance***, ***game condition priority***, ***and diamond's point***. Algoritma greedy by distance diterapkan pada setiap perhitungan keputusan dengan mempertimbangkan jarak terdekat. Sementara itu, game condition priority merupakan serangkaian keputusan if-else dengan urutan prioritas berdasarkan base, teleport, berlian, dan tombol merah. Selain itu, kami juga menerapkan strategi greedy diamonds point, di mana bot akan mengambil berlian dengan poin maksimum untuk meningkatkan efisiensi pengumpulan poin.

Prinsip utamanya adalah memberikan prioritas pada diamond yang terdekat dari base. Strategi ini tidak hanya mempertimbangkan jarak diamond terdekat dari base dan bot, tetapi juga mengkalkulasi jarak diamond terdekat berikutnya yang terdekat dengan diamond terdekat pertama. Dengan demikian, secara

tidak langsung, bot akan bergerak dan menghitung untuk mendapatkan *cluster* diamond terbaik yang memiliki peluang *kerapatan* tertinggi terhadap base.

Selain itu, kami tetap mempertimbangkan perbedaan poin antara diamond biasa dan diamond merah melalui pembobotan terhadap perhitungan awal tersebut. Dengan cara ini, bot tidak hanya mendapatkan banyak diamond, tetapi juga akan lebih cepat menyimpan poin ke dalam base.

Namun, muncul pertanyaan: Bagaimana jika di sekitar base sudah tidak ada lagi diamond yang tersisa? Kami juga mempertimbangkan kondisi di mana sudah tidak ada diamond lagi di sekitar bot dan base. Bot akan bergerak menuju red button untuk memunculkan kembali diamond. Meskipun demikian, ini tidak selalu menguntungkan karena dominasi diamond yang muncul bisa justru di sekitar musuh.

Untuk mengatasi hal ini, kami merasa lebih baik jika bot tetap berada di area yang safe dekat base, menghindari bersaing dengan bot lain di daerah yang jauh dari base. Strategi ini dipicu dengan memperhatikan durasi perjalanan.

Perhitungan jarak pada strategi greedy by distance juga mempertimbangkan fitur teleportasi jika dapat memperpendek jarak yang ingin dituju. Namun, teleportasi kadang-kadang dapat menghalangi rute perjalanan, sehingga kami juga mempertimbangkan untuk menghindari teleportasi agar tidak tersesat.

Selain itu, kami juga tetap mempertimbangkan untuk melakukan parkir jika diamond yang dituju memiliki arah yang sama dan dekat dengan base. Ketika waktu tersisa kurang dari 10 detik, kami akan memprioritaskan untuk segera parkir atau kembali ke base jika membawa diamond.

Dalam mencari diamond, bot kami juga akan mempertimbangkan jika ada bot musuh yang berada diantara diamond yang kita incar (bot musuh dan bot kita dan diamond segaris, yaitu antara rownya sama atau kolomnya sama), dan bot musuh pergi ke arah diamond tersebut, maka dalam hal ini kita akan mencari diamond lain, karena sudah pasti musuh akan sampai ke diamond itu lebih cepat.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

1. Implementasi Algoritma Greedy

Implementasi algoritma greedy pada program bot yang digunakan (pseudocode yang cukup detail dengan komentar untuk pembaca kode agar mudah dipahami).

```
procedure diamond_process()

Algoritma

botPosition = posisi bot sekarang
nearestDiamondBase = posisi diamond terdekat dari bot dan base
redButton = posisi red button
goalPosition = posisi yang akan dituju
if (
   (jarak antar botPosition dan nearestDiamondBase > jarak antar
posisi bot dan redButton) dan (jarak antar botPosition dan
nearestDiamondBase > 2)
)
   goalPosition = redButton
else if((jarak antar botPosition dan nearestDiamondBase) > (jara
antara botPosition dan basePosition)
   goalPosition = basePosition
else
   nearestDiamond = jarak diamond terdekat dari bot saja
   if(jarak botPosition terhadap nearestDiamond <= 2)
      goalPosition = nearestDiamond
   else
      goalPosition = nearestDiamondBase
```

```
function next_move()->tuple(int,int)

Algoritma

teleports_position = list dari semua posisi teleport
red_position = list dari semua posisi red button
enemy_position= list dari semua posisi musuh
diamond_positions = list dari semua posisi diamond
diamonds = list dari game object diamond
base_position = posisi base dari bot kita

Inisialisasi nilai awal
for (obj: semua game object yang ada dalam board)
   if obj.type == "DiamondGameObject":
      # Handle kasus ketika overflow diamond score saat hampir
```

```
penuh oleh red diamond
      if bot.properties.diamonds >= 4 and obj.properties.points ==
2:
          pass
      else:
          diamonds.append(obj)
          diamond_positions.append(obj.position)
            elif obj.type == "TeleportGameObject":
                teleports_position.append(obj.position)
            elif obj.type == "BotGameObject":
                # hanya masukkan posisi musuh
                if bot.properties.name != obj.properties.name:
                    enemy_position.append(obj.position)
            elif obj.type == "DiamondButtonGameObject":
                # masukan redButton ke list
                red_position.append(obj.position)

# teleportPosition berisi array 2 elemen dengan elemen pertama
adalah posisi elemen terdekat terhadap bot dan elemen kedua adalah
counternya

#prosedur swap
if (jarak bot terhadap teleports_position[0] >= jarak bot terhadap
teleports_position[1] ):
            teleports_position[0], teleports_position[1] =
teleports_position[1], teleports_position[0]

#priority 1: kembali ke base
if (waktu tersisa < 10000 and diamond yang dibawa bot > 0) or
diamond yang dibawa bot == 5:
       goal_position = base_position
elif jika jarak bot terhadap enemy <=2:
            goal_position =   self.bot_process(bot, enemy_position,
diamond_positions, diamonds, base_position)

# default prosedur diamond process
else:
       diamond_process(base_position, diamonds, diamond_positions,
bot, red_position[0])

# Kasus menggunakan teleport lebih dekat
if (jarak ke goal position dengan menggunakan teleport < (jarak
antara bot terhadap goal position + jarak goal position terhadap
base))
            goal_position = teleports_position[0]


        if self.goal_position:
           delta_x, delta_y = get_direction_v2(
                bot.position.x,
                bot.position.y,
```

```
                self.goal_position.x,
                self.goal_position.y
            )
           # prosedur untuk menghindari teleport yang tidak
diperlukan akibat fungsi bawaan engine
           curplusportal = Position(x=bot.position.x + delta_x,
y=bot.position.y + delta_y)
            If goal_position != teleports_position[0] and
curplusportal == teleports_position[0]:
                goal_position = dodge_tele(bot.position,
teleports_position[0], teleports_position[1], self.goal_position)
                delta_x, delta_y =
self.get_direction_v2(bot.position.x, bot.position.y,
self.goal_position.x, self.goal_position.y)
                  else:
            # Fungsi bawaan untuk random gerakan ketika goal
position tidak bernilai
            delta = self.directions[self.current_direction]
            delta_x = delta[0]
            delta_y = delta[1]
            if random.random() > 0.6:
                self.current_direction = (current_direction + 1)%
len(directions)

        return delta_x, delta_y
```

```
function bot_process()->Position

Algoritma
curr_pos = posisi bot saat ini
dm_candidate = kandidat diamond terpilih
Enemies_position = list posisi dari musuh

for enemy in enemies_position:
    # mencari arah direction dari enemy terhadap bot
    delta_x_en, delta_y_en = self.get_direction_v2(curr_pos.x,
curr_pos.y, enemy.x, enemy.y)
    # ketika kandidat diamon masih ada
    while len(dm_candidate) > 0:
      nearest_dm = diamond yang terdekat dari base dan bot dari
kandidat diamond
      delta_x_dm, delta_y_dm = self.get_direction_v2(curr_pos.x,
curr_pos.y, nearest_dm.x, nearest_dm.y)
      # jika searah musuh dan diamond awal yang dituju, cari
kandidat nearest yang lain
      if delta_x_en == delta_x_dm and delta_y_en == delta_y_dm:
                  dm_candidate.remove(nearest_dm)
                else:
      # jika tidak searah langsung return sebagai goal position
                  return nearest_dm
```

2. Penjelasan Struktur Data

Struktur data yang kami gunakan dalam membuat logika bot permainan Diamonds terdiri atas Class bernama AlucardGreedy yang disusun oleh empat belas fungsi/prosedur. Berikut tabel fungsi/prosedur yang terdaftar pada Class AlucardGreedy

No.	Nama Fungsi/Prosedur	Deskripsi
1	__init__	Menginisialisasi attribut-attribut Class AlucardGreedy, seperti directions, goal_position, dan current_direction.
2	getDistanceBetween	Mengukur jarak antara dua titik dalam grid berdasarkan jalur horizontal dan/atau vertikal secara ketat
3	getDistanceBetweenTransition	Mengukur jarak antara dua titik dalam grid melalui teleport
4	getNearestObjectPosition	Mencari dan mengembalikan posisi dari suatu objek dalam Board yang terdekat
5	getObjectsInArea	Memberikan list posisi objek dalam game yang berada pada sekitar bot
6	isObjectInArea	Memeriksa apakah terdapat suatu objek game di sekitar area bot
7	isSameDirection	Memeriksa apakah dua posisi target ada pada jalan yang searah
8	get_direction_v2	Mendapatkan arah dari posisi tertentu ke posisi lainnya (North, South, West, atau East)
9	getDistanceWithPortalRelBase	Menghitung jarak total dari posisi bot ke posisi tujuan dengan menggunakan portal dan/atau melewati base
10	dodge_tele	Memberikan posisi tujuan sehingga pergerakan selanjutnya dapat menghindari teleport ke jalan yang salah
11	get_nearest_diamond	Mendapatkan posisi diamond yang terdekat dari posisi bot
12	get_nearest_diamond_base	Mendapatkan posisi diamond yang terdekat dari base dengan mempertimbangkan poin diamond dan jarak dari posisi bot
13	diamond_process	Mencari posisi tujuan yang ingin dicapai dengan mempertimbangkan jarak, kondisi searah, dan jumlah diamond
14	bot_process	Mengecek apakah ada musuh yang berada di antara posisi kita dengan target diamond (segaris) , jika ada, maka kita akan mencari diamond lain selain diamond tersebut

15	next_move	Memutuskan untuk melangkah ke arah mana (North, South, West, atau East)

3. Analisis dari Desain Solusi Algoritma Greedy

Hal yang Diuji	Implementasi Strategi	Hasil
Pengambilan diamond	Strategi diimplementasikan dengan mengutamakan diamond yg relatif dekat terhadap bot dan base, terdekat dengan diamond berikutnya, sekaligus diamond dengan poin yang lebih besar	Strategi berjalan dengan sangat baik sebagai *main power* dari algoritma kita.
Mencari alternatif diamond lain jika telah didahului/dihadang musuh	Strategi diimplementasikan untuk mengatasi kasus mengejar diamond yang sama dengan tujuan musuh, dimana bot kita kalah posisi di suatu kolom/baris dengan cara mencari nearest diamond lain, strategi ini aktif ketika musuh berada pada jarak <=2 dari bot	Strategi berjalan dengan baik, kecuali saat inventory musuh penuh dan mereka secara kebetulan menghadang jalur menuju diamond.
Menginjak red button	Strategi ini digunakan jika objek terdekat adalah red button dan tidak ada diamond dengan jarak kurang dari 3 dari bot kami	Strategi berjalan dengan baik
Menggunakan teleporter untuk memperpendek jarak	Strategi ini hanya digunakan untuk mereplace goal position yang telah di set sebelumnya, program tidak akan iterasi seluruh diamond untuk mengecek jarak, hal ini karena kami mengandalkan algoritma pengambilan diamond kami sebagai main power	Strategi berjalan dengan baik
Menghindari teleporter	Strategi ini digunakan ketika teleporter menghalangi jalan untuk menuju posisi tujuan, dan bot tidak ingin masuk teleporter maka bot akan mengambil rute memutar agar tidak masuk ke teleporter dengan tidak sengaja	Strategi berjalan dengan baik, kecuali ketika ada dua teleporter yang bersebelahan, maka bot akan memutar bolak-balik karena ingin menghindari kedua teleporter
Kembali ke base ketika waktu yang tersisa telah mendekati akhir	Jika waktu sudah kurang dari 10 detik dan diamond di inventory > 1, maka bot akan kembali ke base	Strategi berjalan baik, kecuali ketika base terlalu jauh maka ada kemungkinan bot tidak mencapai base ketika game berakhir
Kembali ke base jika	Strategi ini membuat bot kami jalan ke base /	Strategi berjalan baik

inventory penuh	sambil melewati teleporter atau jika intendednya adalah tidak melewati teleporter, maka bot akan menghindar dari teleporter	
Melewati base terlebih dahulu sebelum menuju ke diamond yang searah dengan base	Strategi ini digunakan jika base lebih dekat daripada diamond target tujuan dan base searah dengan diamond, maka bot akan ke base terlebih dahulu untuk menyetor diamond	Strategi berjalan baik

BAB V
KESIMPULAN DAN SARAN

1. Kesimpulan

Kesimpulan yang dapat diambil oleh kelompok kami adalah bahwa algoritma greedy, yang fokus pada pemilihan solusi lokal terbaik pada setiap langkah, dapat efisien dalam pengambilan keputusan yang cepat dan memberikan solusi yang baik secara lokal. Meskipun demikian, penting untuk diingat bahwa solusi lokal terbaik tidak selalu menghasilkan solusi global terbaik. Algoritma greedy cocok untuk keadaan di mana kecepatan eksekusi menjadi prioritas, seperti dalam pembuatan bot, tetapi perlu hati-hati dalam situasi di mana keputusan setiap langkah dapat berdampak signifikan pada hasil akhir. Dalam beberapa konteks yang lebih kompleks, pendekatan yang lebih cermat dan komprehensif mungkin diperlukan untuk mencapai solusi global optimal.

2. Saran

Saran untuk kelompok kami diantaranya adalah agar lebih matang lagi dalam perencanaan pembuatan program, lebih memperhatikan spesifikasi penting dari tugas besar, dan lebih banyak sesi *brainstorming* diantara sesi pengerjaan.

LAMPIRAN

Pranala Youtube : https://youtu.be/9MXRM4zggRI?si=zADk3sNxqKR4i7DO

Pranala Github : https://github.com/fairuzald/Tubes1_Alucard/

DAFTAR PUSTAKA

Munir, Rinaldi. Homepage Rinaldi Munir. Diakses dari

https://informatika.stei.itb.ac.id/~rinaldi.munir/

Etimo dkk. Diakses dari

https://github.com/Etimo/diamonds