LAPORAN TUGAS BESAR 1
IF2211 STRATEGI ALGORITMA

Pemanfaatan Algoritma *Greedy* dalam Pembuatan Bot Permainan Diamonds

Disusun oleh:

Agil Fadillah Sabri	13522006
Bastian H. Suryapratama	13522034
Haikal Assyauqi	13522052

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

DAFTAR ISI

DAFTAR GAMBAR

BAB I
DESKRIPSI TUGAS

Diamonds merupakan suatu *programming challenge* yang mempertandingkan bot yang dibuat oleh tim sendiri dengan bot dari para pemain lainnya. Setiap pemain akan memiliki sebuah bot di mana 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 botnya. Penjelasan lebih lanjut mengenai aturan permainan akan dijelaskan di bawah.

Gambar 1. Tampilan Laman Utama Permainan Diamonds

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.

Program permainan Diamonds terdiri atas:

1. *Game engine*, yang secara umum berisi:
   a. Kode *backend* permainan, yang berisi *logic* permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan *frontend* dan program bot.
   b. Kode *frontend* permainan, yang berfungsi untuk memvisualisasikan permainan.

2. *Bot starter pack*, yang secara umum berisi:
    a. Program untuk memanggil API yang tersedia pada *backend*.
    b. Program *bot logic* (bagian ini yang akan diimplementasikan dengan algoritma *greedy* untuk bot tiap kelompok).
    c. Program utama (*main*) dan utilitas lainnya.

Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan *game engine* dan membuat bot dari *bot starter pack* yang telah tersedia pada pranala berikut.

a. *Game engine*:
   https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0
b. *Bot starter pack*:
   https://github.com/haziqam/tubes1-IF2211-bot-starter-pack/releases/tag/v1.0.1

Komponen-komponen dari permainan Diamonds antara lain:

1. *Diamonds*

Gambar 2. Logo *Diamond*

Untuk memenangkan pertandingan, pemain 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*

Gambar 3. Logo *Diamond Button*

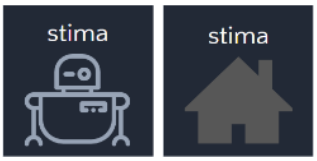
Ketika *red button* ini dilewati/dilangkahi, semua *diamond* akan di-*generate* kembali pada *board* dengan posisi acak. Posisi *red button* ini juga akan berubah secara acak jika *red button* ini dilangkahi.

3. *Teleporters*

Gambar 4. Logo *Teleporter*

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

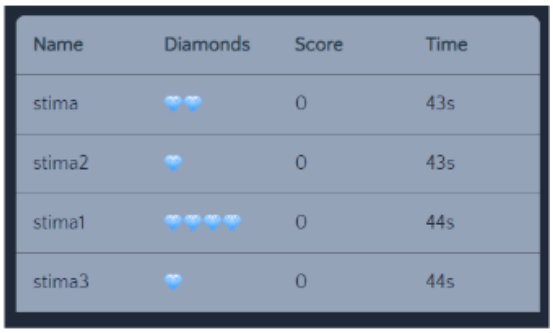
4. *Bots and Bases*



Gambar 5. Logo Bot dan *Base*

Pada game ini pemain 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*



Gambar 6. Data *Inventory* Bot

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 pemain 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

2.1 Dasar Teori Algoritma *Greedy*

2.1.1 Definisi Algoritma Greedy

Algoritma *greedy* adalah sebuah teknik dalam pemrograman yang digunakan untuk menyelesaikan masalah optimasi atau pengambilan keputusan. Pendekatan ini didasarkan pada prinsip mengambil keputusan lokal yang optimal pada setiap tahap atau langkahnya, dengan harapan menghasilkan solusi global yang optimal secara keseluruhan.

Pada dasarnya, algoritma *greedy* berusaha mencari solusi terbaik dari suatu masalah dengan mempertimbangkan pilihan yang tersedia pada setiap langkahnya, dan memilih pilihan yang paling optimal pada saat itu, tanpa memperhitungkan konsekuensi jangka panjang dari pilihan tersebut.

Contoh penerapan algoritma *greedy* adalah dalam masalah pemilihan koin. Jika seseorang ingin memberikan kembalian untuk suatu jumlah uang dengan jumlah koin yang minimal, maka algoritma *greedy* akan memilih koin dengan nilai tertinggi yang masih bisa digunakan untuk memberikan kembalian, sampai jumlah kembalian yang diberikan sesuai dengan jumlah uang yang diminta. Meskipun algoritma *greedy* ini tidak selalu menghasilkan solusi yang optimal secara keseluruhan, namun seringkali algoritma ini menghasilkan solusi yang cukup baik dan efisien.

2.1.2 Kelebihan dan Kekurangan Algoritma *Greedy*

Algoritma *greedy* digunakan untuk menyelesaikan masalah optimasi atau pengambilan keputusan dengan cara mengambil keputusan lokal yang optimal pada setiap langkahnya, dengan harapan menghasilkan solusi global yang optimal secara keseluruhan. Namun algoritma *greedy* ini memiliki beberapa kelebihan dan kekurangan. Adapun beberapa kelebihan algoritma *greedy* antara lain:

1. Solusi Cukup Efisien

   Algoritma *greedy* seringkali menghasilkan solusi yang cukup efisien, terutama dari segi waktu dan sumber daya komputasi yang digunakan, karena pada setiap langkahnya hanya mempertimbangkan pilihan yang optimal pada saat itu tanpa memperhitungkan konsekuensi jangka panjang. Dalam beberapa kasus, algoritma *greedy* bahkan mampu

menghasilkan solusi yang optimal secara keseluruhan, sehingga sangat berguna dalam menyelesaikan masalah dengan waktu yang terbatas.

2. Sederhana dan Mudah Diimplementasikan

   Algoritma *greedy* sangat sederhana dan mudah diimplementasikan, karena hanya melibatkan proses pengambilan keputusan sederhana pada setiap langkahnya. Oleh karena itu, algoritma *greedy* cocok digunakan dalam kasus-kasus di mana kompleksitas waktu dan ruang menjadi masalah yang besar.

3. Mudah Diterapkan pada Berbagai Jenis Masalah

   Algoritma *greedy* dapat diterapkan pada berbagai jenis masalah, seperti masalah pengoptimalan, pengambilan keputusan, dan masalah grafik. Dalam masing-masing kasus, algoritma *greedy* memerlukan strategi yang berbeda untuk memilih pilihan yang optimal pada setiap langkahnya.

Namun, algoritma *greedy* juga memiliki beberapa kekurangan, diantaranya:

1. Tidak Selalu Menghasilkan Solusi Optimal

   Walaupun algoritma *greedy* dapat menghasilkan solusi yang efisien, namun tidak selalu menghasilkan solusi yang optimal secara keseluruhan. Pada beberapa kasus, algoritma *greedy* dapat menghasilkan solusi yang suboptimal atau bahkan tidak dapat menyelesaikan masalah yang diberikan. Oleh karena itu, sebelum menggunakan algoritma *greedy*, diperlukan analisis dan perhitungan yang cermat untuk menentukan apakah algoritma ini cocok untuk digunakan pada kasus tertentu atau tidak.

2.1.3 Prinsip Algoritma Greedy

Prinsip algoritma *greedy* adalah memilih pilihan yang optimal pada setiap langkah, dengan harapan akan menghasilkan solusi global yang optimal secara keseluruhan. Dalam prakteknya, hal ini berarti bahwa pada setiap langkah, algoritma *greedy* akan memilih opsi yang terbaik berdasarkan kriteria yang diberikan, tanpa mempertimbangkan konsekuensi jangka panjang dari opsi tersebut. Beberapa prinsip penting dari algoritma *greedy* antara lain:

1. *Greedy-choice Property*

   Prinsip *greedy-choice property* adalah dasar dari algoritma *greedy*. Prinsip ini menyatakan bahwa pada setiap langkah, algoritma *greedy* akan memilih pilihan yang terbaik dari segi lokal, tanpa mempertimbangkan solusi global secara keseluruhan. Artinya, algoritma *greedy* akan memilih opsi yang memberikan hasil terbaik pada saat itu, tanpa

mempertimbangkan konsekuensi jangka panjang. Prinsip ini berlaku untuk hampir semua kasus penggunaan algoritma *greedy*, seperti memilih koin-koin untuk memberikan kembalian atau memilih rute terpendek pada grafik.

2. *Optimal Substructure*

   Prinsip optimal *substructure* adalah prinsip yang menyatakan bahwa solusi optimal untuk sebuah masalah dapat dicapai melalui kombinasi solusi optimal untuk submasalah yang lebih kecil. Dalam konteks algoritma *greedy*, prinsip ini sering digunakan untuk mempercepat proses pencarian solusi optimal dengan mengurangi jumlah opsi yang perlu dievaluasi. Dengan mengikuti prinsip *optimal substructure*, algoritma *greedy* dapat mempercepat proses pencarian solusi dan menghindari evaluasi opsi yang tidak relevan.

3. Proses *Greedy*

   Proses *greedy* adalah proses yang dilakukan oleh algoritma *greedy* untuk memilih opsi terbaik pada setiap langkah. Proses ini dapat berbeda-beda tergantung pada masalah yang dihadapi, tetapi prinsipnya adalah sama: algoritma *greedy* akan mempertimbangkan semua opsi yang tersedia pada setiap langkah, dan memilih opsi yang memberikan hasil terbaik pada saat itu. Setelah opsi terbaik dipilih, algoritma *greedy* akan memperbarui solusi sebelum melangkah ke langkah berikutnya.

4. Analisis Kebenaran Algoritma *Greedy*

   Algoritma *greedy* tidak selalu menghasilkan solusi optimal. Oleh karena itu, sebelum menggunakan algoritma *greedy* pada suatu masalah, diperlukan analisis yang cermat untuk menentukan apakah algoritma ini benar-benar cocok untuk digunakan pada masalah tersebut. Analisis ini melibatkan pengecekan apakah masalah memiliki *greedy-choice property* dan optimal *substructure*, serta apakah algoritma *greedy* dapat menghasilkan solusi yang benar dalam semua kasus.

2.1.4 Elemen-Elemen *Greedy*

1. Himpunan Kandidat

   Himpunan kandidat (C) berisi kandidat solusi yang akan dipilih pada setiap langkah.

2. Himpunan Solusi

   Himpunan solusi (S) berisi kandidat yang terpilih sebagai solusi dari permasalahan.

3. Fungsi Solusi

   Fungsi solusi digunakan untuk menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.

4. Fungsi Seleksi (*Selection Function*)

   Fungsi seleksi berfungsi memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah terpilih pada langkah sebelumnya tidak perlu dipertimbangkan lagi pada langkah berikutnya.

5. Fungsi Kelayakan (*Feasible*)

   Fungsi kelayakan mengevaluasi apakah kandidat yang terpilih dapat memberikan solusi yang layak, yaitu kandidat tersebut dan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi.

6. Fungsi Objektif

   Fungsi objektif menentukan apakah permasalahan membutuhkan memaksimumkan atau meminimumkan solusi

2.2 Cara Kerja Program

Permainan Diamonds adalah permainan yang berbasis web. Ada 2 *building blocks* utama dari permainan ini, yaitu *game engine* dan *logic* untuk bot.

Game engine adalah bagian yang mengatur keberjalanan *game*, mulai dari peletakkan objek-objek di dalam *map*, mengatur batasan terhadap langkah-langkah bot yang dijalankan, sampai perhitungan *score* yang didapatkan oleh masing-masing bot.

Logic adalah cara suatu bot dalam menentukan langkah-langkah yang akan diambil. Semua aksi yang diambil oleh suatu bot berasal dari *logic* yang dipakai oleh bot tersebut.

Sebenarnya, sudah tersedia *logic* bawaan yang terdapat di dalam file random.py. Akan tetapi, kita bisa membuat logic sendiri. Cara untuk mengimplementasikan logic yang kita buat sendiri adalah:

1. Buat file .py baru di dalam folder *logic*. Folder *logic* berada di dalam folder tubes1-IF2211-bot-starter-pack-1.0.1/game (diasumsikan nama *root* folder dari bot adalah tubes1-IF2211-bot-starter-pack-1.0.1).
2. Buat suatu *class* baru dengan menurunkan kelas dasar yang bernama BaseLogic. Kelas BaseLogic berada di dalam file base.py (masih di dalam folder *logic*).
3. Buat 2 *method* utama yang akan dijalankan oleh bot, yaitu __init__ dan next_move. Contoh kedua method tersebut terdapat di dalam *class* RandomLogic yang berada di dalam file random.py (masih di dalam folder logic).
4. Modifikasi file .py yang baru dibuat sesuai dengan algoritma yang diinginkan.

Sebelum menjalankan *logic* yang sudah dibuat, kita perlu mendaftarkan *logic* tersebut ke dalam file main.py yang berada di dalam folder tubes1-IF2211-bot-starter-pack-1.0.1. Di dalam file tersebut, pertama-tama kita perlu melakukan *import* file *logic* yang telah dibuat. Setelah itu, masukkan nama *logic* (nama bebas) dan *class* logic yang kita buat ke dalam *variable dictionary* CONTROLLERS.

Setelah mendaftarkan *logic*, kita perlu mengubah isi file run-bots.bat atau run-bots.sh untuk mengatur segala informasi mengenai bot yang akan dijalankan. Kita dapat mengatur *email*, nama, *password*, dan nama team sesuai keinginan kita asalkan merupakan input yang valid. Untuk *logic*, kita perlu memasukkan nama *logic* yang telah didaftarkan di dalam file main.py.

Untuk menjalankan bot, kita perlu menjalankan game engine terlebih dahulu. Langkah-langkah menjalankan *game engine* adalah sebagai berikut.

1. Masuk ke *root* folder *game engine*.
   ```
   cd tubes1-IF2110-game-engine-1.1.0
   ```
2. Install *dependencies* menggunakan yarn.
   ```
   yarn
   ```
3. *Setup default environment variable* dengan menjalankan *script* berikut.
   - Untuk Windows:
     ```
     ./scripts/copy-env.bat
     ```
   - Untuk Linux/(*possibly*) MacOS:
     ```
     chmod +x ./scripts/copy-env.sh
     ./scripts/copy-env.sh
     ```
4. *Setup local database* (buka aplikasi docker desktop terlebih dahulu, lalu jalankan *command* berikut di terminal).
   ```
   docker compose up -d database
   ```
   - Lalu jalankan script berikut. Untuk Windows:
     ```
     ./scripts/setup-db-prisma.bat
     ```
   - Untuk Linux / (possibly) macOS:
     ```
     chmod +x ./scripts/setup-db-prisma.sh
     ./scripts/setup-db-prisma.sh
     ```
5. Build.
   ```
   npm run build
   ```

6. Run.
   ```
   npm run start
   ```
7. Kunjungi *frontend* melalui http://localhost:8082/.

Setelah *set-up game engine* dilakukan, kita dapat menjalankan bot dengan *logic* yang sudah kita buat dengan cara sebagai berikut.

1. Masuk ke *root* directory dari *project* (sesuaikan dengan nama rilis terbaru).
   ```
   cd tubes1-IF2110-bot-starter-pack-1.0.1
   ```
2. Install dependencies menggunakan pip
   ```
   pip install -r requirements.txt
   ```
3. Run
   - Untuk windows:
     ```
     ./run-bots.bat
     ```
   - Untuk Linux / (possibly) macOS:
     ```
     ./run-bots.sh
     ```

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 *Mapping* Persoalan Diamonds

1. Himpunan Kandidat : Himpunan semua *diamond* yang ada pada papan permainan.
2. Himpunan Solusi : *Diamond* yang terpilih.
3. Fungsi Solusi : Memeriksa apakah *inventory* saat ini sudah berisi 5 poin atau belum.
4. Fungsi Seleksi (*Selection Function*) : Disesuaikan dengan strategi yang dipakai (dijelaskan lebih lanjut pada subbab **3.2 Eksplorasi Alternatif Solusi *Greedy***).
5. Fungsi Kelayakan (*Feasible*) : Memeriksa apakah kandidat *diamond* yang dipilih bisa dimasukkan ke dalam *inventory* (total *inventory* tidak boleh lebih dari 5 poin).
6. Fungsi Objektif : Poin yang diperoleh maksimum.

3.2 Eksplorasi Alternatif Solusi *Greedy*

Terdapat 3 solusi yang kami ajukan untuk dijadikan sebagai sebagai strategi program untuk menjalankan bot:

3.2.1 *Greedy by Point*

Greedy by Points adalah pendekatan *greedy* pada permainan Diamonds yang memprioritaskan *diamond* yang memiliki poin lebih besar, yaitu *diamond* merah.

Mapping elemen *greedy*:

1. Himpunan Kandidat: Himpunan semua *diamond* merah.
2. Himpunan solusi: *Diamond* merah yang terpilih.
3. Fungsi solusi: Memeriksa apakah *inventory* saat ini sudah berisi ≥ 4 poin atau belum.
4. Fungsi seleksi: Pilihlah *diamond* merah terdekat jika ada, jika tidak bergerak menuju *diamond button*.
5. Fungsi kelayakan: Memeriksa apakah *diamond* yang dipilih dapat dimasukkan ke dalam *inventory*.
6. Fungsi objektif: Mengambil *diamond* merah sebanyak mungkin.

3.2.2 *Greedy by Position*

Greedy by Position adalah pendekatan *greedy* pada permainan Diamonds yang hanya melihat *diamond* yang berada pada satu baris atau satu kolom dengan posisi bot saat ini. Selain itu, jika terdapat *teleporter* yang berada pada satu baris/kolom dengan posisi bot saat ini, akan dianggap sebagai perpanjangan baris/kolom bot, sehingga *diamond* yang berada satu baris/kolom dengan *teleporter* yang lain juga akan diperhitungkan.

Pada strategi ini, akan diprioritaskan terlebih dahulu *diamond* merah. Jika tidak ada *diamond* merah yang memenuhi syarat, maka baru dicari *diamond* biru. Jika terdapat dua/lebih *diamond* dengan poin yang sama, maka akan dipilih *diamond* yang memiliki jarak terdekat atau jumlah langkah terkecil dari posisi bot saat ini.

Mapping elemen *greedy*:

1. Himpunan Kandidat: Himpunan semua *diamond* yang berada satu baris/kolom dengan bot, himpunan semua *diamond* yang berada satu baris/kolom dengan *teleporter* kedua jika terdapat *teleporter* yang satu baris/kolom dengan bot.
2. Himpunan solusi: *Diamond* yang terpilih.
3. Fungsi solusi: Memeriksa apakah *inventory* saat ini sudah berisi 5 poin atau belum.
4. Fungsi seleksi: Pilihlah *diamond* merah terdekat yang berada satu baris/kolom dengan bot atau teleporter jika ada, jika tidak pilihlah *diamond* biru yang berada satu baris/kolom dengan bot atau teleporter jika ada, jika tidak ada bergerak menuju *diamond button*.
5. Fungsi kelayakan: Memeriksa apakah *diamond* yang dipilih dapat dimasukkan ke dalam *inventory*.
6. Fungsi objektif: Poin yang diperoleh setinggi mungkin.

3.2.3 *Greedy by Ratio*

Greedy by ratio adalah pendekatan *greedy* pada permainan Diamonds yang memperhitungkan nilai rasio dari 3 komponen penting dalam menghasilkan poin yaitu, *blue diamond*, *red diamond*, dan *diamond button*. Rasio yang digunakan yaitu perbandingan antara jarak antara posisi bot dengan komponen dan nilai pembagi komponen. Tiap komponen memiliki masing-masing nilai pembagi, yaitu *red diamond* memiliki poin 2 (sesuai dengan nilai poin untuk *diamond* merah), *blue diamond* memiliki poin 1 (sesuai dengan nilai poin untuk *diamond* biru), dan *diamond button* diberikan poin 0,8.

Alasan pemilihan poin 0,8 untuk *diamond button* adalah karena anggapan bahwa *diamond button* memiliki prioritas lebih rendah untuk dipilih dibandingkan *diamond*, mengingat objektif permainan ini adalah untuk mendapatkan *diamond* sebanyak-banyaknya. Dengan angka tersebut, *diamond button* baru akan dipilih sebagai tujuan pergerakan bot ketika jarak *diamond* terdekat ke bot terlalu jauh atau ketika tidak ada sama sekali *diamond* di papan permainan.

Selain itu, diperhatikan juga posisi bot pemain lain untuk menghindari terjadinya *tackle* oleh pemain lawan. Jika jarak bot milik sendiri lebih dekat daripada jarak bot pemain lain ke *diamond* yang dituju saat ini, maka bot tetap akan bergerak menuju *diamond* tersbeut. Jika sebalikanya yang terjadi, maka *diamond* tersebut diabaikan dan dipilih *diamond* lain dengan rasio terkecil kedua.

Mapping elemen greedy:

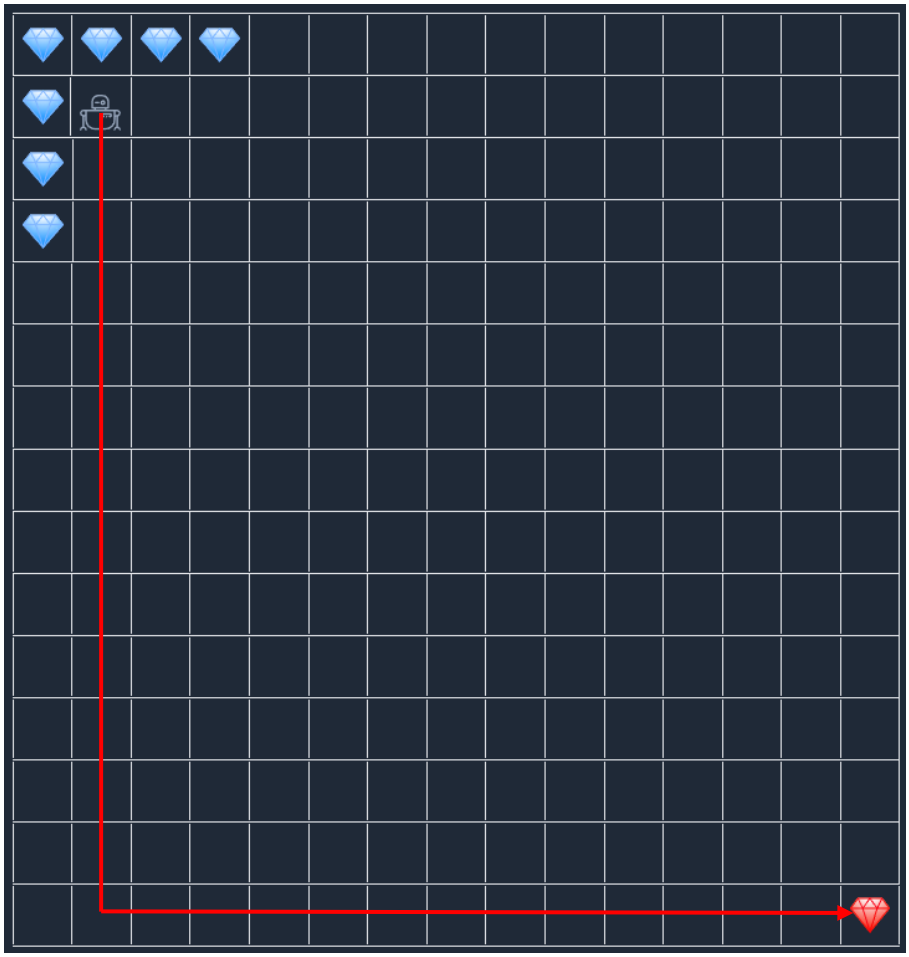
1. Himpunan Kandidat: Himpunan *diamond* yang terdapat di seluruh papan permainan.
2. Himpunan solusi: *Diamond* dengan nilai rasio (jumlah langkah / poin) terkecil.
3. Fungsi solusi: Memeriksa apakah *inventory* saat ini sudah berisi 5 poin atau belum.
4. Fungsi seleksi: Pilih objek yang memiliki rasio paling kecil.
5. Fungsi kelayakan: Memeriksa apakah *diamond* yang dipilih dapat dimasukkan ke dalam *inventory*.
6. Fungsi objektif: Poin yang diperoleh setinggi mungkin.

3.3 Analisis Efisiensi dan Efektivitas

3.3.1 *Greedy by Point*

Pada strategi *greedy by point*, hanya *diamond* merah yang akan dilihat sebagai kandidat solusi oleh bot. Akibatnya, strategi ini akan kurang efektif dan efisien jika seandainya terdapat banyak *diamond* biru di sekitar bot, tetapi di saat yang sama juga terdapat *diamond* merah yang letaknya sangat jauh dari bot, karena bot akan bergerak menuju diamond merah tersebut, bukan mengambil diamond biru yang ada di sekitarnya yang berjarak lebih dekat.

Adapun ilustrasinya sebagai berikut:

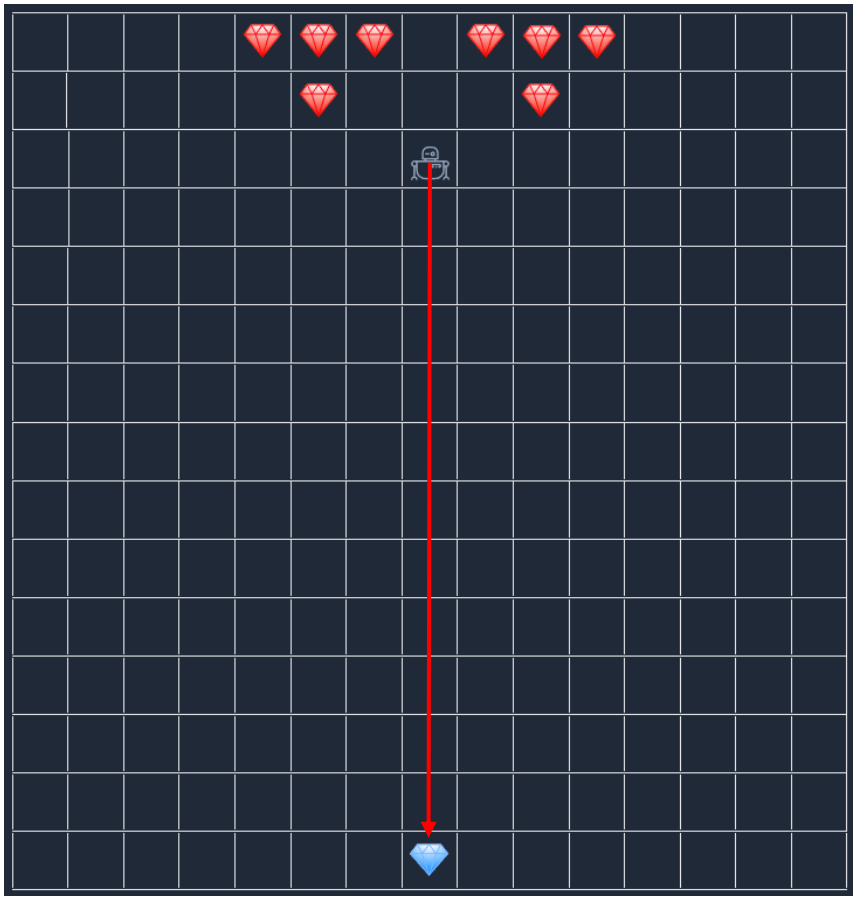


Gambar 7. Ilustrasi Ketidakoptimalan *Greedy by Point*

3.3.2 *Greedy by Position*

Pada strategi *greedy* by position ini, hanya *diamond* yang berposisi satu baris/satu kolom dengan bot yang akan diperhatikan. Akibatnya, strategi ini akan kurang efektif dan efisien jika seandainya terdapat banyak *diamond* di sekitar bot, tetapi posisinya tidak satu baris/kolom dengan bot, dan disaat yang sama, terdapat satu *diamond* yang satu baris/kolom dengan bot tetapi jaraknya sangat jauh dengan bot, karena bot akan bergerak menuju *diamond* yang jauh tersebut, bahkan walau poinnya lebih kecil.

Adapun ilustrasinya sebagai berikut:



Gambar 8. Ilustrasi Ketidakoptimalan *Greedy by Position*

3.3.3 *Greedy by Ratio*

Pada strategi *greedy by ratio*, akan melihat seluruh diamond yang ada di dalam papan permainan sebagai bagian dari kandidat solusi, sehingga dapat dikatakan strategi ini cukup sangkil dan mangkus, terutama dibandingkan dua algoritma sebelumnya. Namun, algoritma ini tidak memperhatikan *teleport*, karena *teleport* pada permainan ini bersifat dinamis sehingga pergerakan menggunakan *teleport* menjadi tidak terprediksi.

3.4 Strategi *Greedy* yang Dipilih

Berdasarkan hasil analisis yang telah dilakukan pada subbab sebelumnya, maka terpilihlah strategi *Greedy by Ratio* sebagai algoritma untuk menggerakkan bot, karena dinilai paling sangkil dan mangkus dibandingkan dua strategi lainnya. Hal ini karena dengan strategi *greedy by ratio*, semua *diamond* yang ada pada papan permainan akan dimasukkan sebagai kandidat solusi, sehingga memperluas kemungkinan *diamond* yang terpilih dan tidak akan melewatkan *diamond-diamond* yang berjarak sangat dekat dengan bot, yang berbeda dengan dua algoritma lainnya, yang hanya mampu melihat sebagian *diamond* sebagai kandidat solusi.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma *Greedy*

```
class Greedy_by_Ratio(BaseLogic):
{Kelas yang mengimplementasikan Algoritma Greedy yang memilih diamond berdasarkan rasio (jarak : point). Bot akan memilih diamond yang memiliki rasio (jarak : point) terkecil}
```

```
Atribut:
self.directions : [(1, 0), (0, 1), (-1, 0), (0, -1)]
self.goal_position: Optional[Position] = None
self.current_direction = 0
```

```
Method:
function next_move(self, board_bot: GameObject, board: Board)
     → (int, int)
{Menentukan arah gerak bot selanjutnya}

Deklarasi:
   props : Properties { GameObject.properties }
   base : Base { GameObject.properties.base }
   delta_x, delta_y : int

Algoritma:
   { Mencari diamond sengan rasio terkecil }
   self.goal_position ← search_diamond_with_minimum_ratio(
                          board_bot, board)
   { Mendapatkan properti bot }
   props ← board_bot.properties

   { MENENTUKAN KONDISI BOT SAAT INI }
   if props.diamonds = 5 then
      base ← board_bot.properties.base
      self.goal_position ← base
   elif props.diamonds >= 3 and props.milliseconds_left < 12000 then
      { 12 detik terakhir, harus kembali ke base }
      self.goal_position ← board_bot.properties.base
```

```
{ MENENTUKAN DELTA_X DAN DELTA_Y }
current_position ← board_bot.position
if self.goal_position then
   { Bergerak ko objek yang spesifik, hitung delta }
   delta_x, delta_y ← get_direction(
             current_position.x,
             current_position.y,
             self.goal_position.x,
             self.goal_position.y,
         )
else
   { Bergerak ke sekitar }
   delta ← self.directions[self.current_direction]
   delta_x ← delta[0]
   delta_y ← delta[1]
   if random.random() > 0.6 then
      self.current_direction ← (self.current_direction + 1) %
                               len(self.directions)

{ Menghindari invalid move }
if (delta_x = 0 and delta_y = 0) then
   delta_y ← 1
elif (delta_x = 1 and delta_y = 1) then
   delta_y ← 0
elif (delta_x = -1 and delta_y = -1) then
   delta_y ← 0

return delta_x, delta_y
```

```
function search_diamond_with_minimum_ratio (
        board_bot: GameObject, board: Board) -> Position
{ Mengembalikan posisi diamond yang memiliki rasio (jarak : point)
terkecil jika ada, mengembalikan posisi diamond button jika tidak
ada diamond yang tersedia }

Deklarasi:
   Object_Position, diamond_button : Position
   Min_ratio, delta_x, delta_y : int
   jumlah_langkah, langkah_musuh : int
   ratio_diamond : real

Algoritma:
   Object_Position ← Position
   min_ratio ← 9999

   { Memilih untuk mendekati tombol reset terlebih dahulu }
   diamond_button ← get_diamond_button(board)
   delta_x ← abs(board_bot.position.x - diamond_button.x)
   delta_y ← abs(board_bot.position.y - diamond_button.y)
   ratio_diamond ← (delta_x + delta_y) / 0.8
   if ratio_diamond < min_ratio then
      Object_Position ← diamond_button
      min_ratio ← ratio_diamond

   { Mencari diamond yang memiliki rasio terkecil }
   for i in range(len(board.diamonds)) do
      delta_x ← abs(board_bot.position.x -
                    board.diamonds[i].position.x)
      delta_y ← abs(board_bot.position.y -
                    board.diamonds[i].position.y)
      jumlah_langkah ← delta_x + delta_y
      ratio_diamond ← jumlah_langkah /
                      board.diamonds[i].properties.points
```

```
      { Menghindari bertabrakan dengan musuh }
      langkah_musuh ← the_closest_enemy_to_the_diamond(board,
                      board.diamonds[i].position)
      if (ratio_diamond < min_ratio and
         jumlah_langkah <= langkah_musuh) then
         if not (board.diamonds[i].properties.points = 2 and
                  board_bot.properties.diamonds = 4) then
                min_ratio ← ratio_diamond
                Object_Position ← board.diamonds[i].position

   return Object_Position
```

```
function get_diamond_button(board: Board) -> Position
{ Mengembalikan posisi diamond button }
Deklarasi :
Algoritma :
   for i in range (len(board.game_objects)) do
      if (board.game_objects[i].type =
          'DiamondButtonGameObject') then
         return board.game_objects[i].position
```

```
function the_closest_enemy_to_the_diamond(board : Board,
         diamondposition : Position) -> int :
{ Mengembalikan jarak musuh terdekat ke diamond yang kita tuju }
Deklarasi :
   step_musuh, x, y : int
Algoritma :
   step_musuh ← 9999
   for i in range(len(board.bots)) then
      x ← abs(board.bots[i].position.x - diamondposition.x)
      y ← abs(board.bots[i].position.y - diamondposition.y)
      if (step_musuh > (x+y)) then
         step_musuh ← x+y
   return step_musuh
```

4.2 Struktur Data yang Digunakan

Pada pengimplementasian algoritma *greedy* di atas, kami menggunakan beberapa struktur data sebagai berikut:

1. *Enumeration*

   Tipe data ini digunakan untuk mendaftarkan seluruh pergerakan bot yang diperbolehkan. Dilakukan dengan mendaftarkan secara langsung tiap-tiap pergerakan yang diizinkan ke dalam sebuah array. Tipe data ini digunakan sebagai atribut dari **class Greedy_by_Point**.

   Pengimplementasian :

   ```
   self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
   ```

2. *List*

   Tipe data ini digunakan untuk menyimpan seluruh objek-objek yang ada pada permainan. Berikut adalah bentuk-bentuk pengimplementasiannya:

   a. *List of Diamonds*

      Digunakan untuk mendaftarkan seluruh *diamond* yang tersedia di dalam permainan. Berguna dalam mencari *diamond* mana yang akan dituju oleh bot.

   b. *List of Bots*

      Digunakan untuk mendaftarkan seluruh bot dari pemain lawan. Berguna dalam menentukan langkah penghindaran dari mekanisme *tackle*.

   c. *List of GameObject*

      Digunakan untuk mendaftarkan seluruh objek-objek game yang ada di dalam permainan. Berguna dalam melakukan pencarian *diamond button*.

4.3 Analisis Desain Solusi Algoritma *Greedy*

Dalam algoritma yang dibuat, kami mendefinisikan beberapa kasus pada setiap permainan. Yang pertama adalah kami menentukan perintah yang akan dieksekusi menggunakan sistem rasio. Namun ketika bot berada pada kondisi yang kurang menguntungkan, seperti:

1. *Base* yang berada di ujung papan permainan.
2. Terdapat musuh yang mendekat.
3. Jarak diamond yang terlalu jauh.

Maka jika terdapat kondisi ataupun beberapa kondisi di atas sekaligus, maka kami akan mengarahkan bot menuju *diamond button*. Hal ini bertujuan agar selain mengubah keadaan bot milik sendiri agar lebih menguntungkan, juga akan mempersulit pergerakan bot lawan akibat *generate diamond* secara tiba-tiba akibat pemencetan *diamond button.*

Adapun jika bot sedang tidak berada pada kondisi-kondisi di atas, maka bot akan memilih *diamond*, baik *red diamond* maupun *blue diamond*, tergantung rasio masing-masing *diamond* tersebut.

BAB V
KESIMPULAN

5.1 Simpulan

Algoritma *greedy* dapat digunakan untuk menemukan solusi optimum lokal berdasarkan kondisi saat ini. Akan tetapi, solusi optimum lokal yang didapatkan berdasarkan kondisi tertentu belum tentu akan menjadi solusi optimum global yang berlaku di segala kondisi. Berdasarkan pengalaman kami dalam pembuatan *logic* bot dengan algoritma *greedy*, algoritma ini cukup baik untuk digunakan dalam proses pengambilan keputusan oleh bot. Dalam kondisi-kondisi yang umum, pengambilan solusi optimum lokal memberikan hasil yang cukup memuaskan dan mendekati optimum global. Solusi tersebut hanya akan memberikan hasil yang kurang baik ketika terjadi kondisi-kondisi spesifik yang cukup jarang terjadi.

5.2 Saran

Beberapa saran dari kelompok kami di antaranya:

1. Lebih ditingkatkan lagi efektivitas, efisiensi, serta *readability* dari algoritma bot yang telah dibuat.
2. Pahami cara kerja program secara lebih mendalam, karena terdapat mekanisme *game* yang bisa dijadikan ide dalam pembuatan algoritma *greedy*, contohnya *tackle*. Mekanisme tersebut dapat dimanfaatkan untuk mengembangkan bot yang berfokus untuk mencuri *diamond* dari bot lawan. Algoritma *greedy* yang kami buat belum menyertakan mekanisme tersebut.
3. Usahakan pengerjaaan tugas besar lebih terarah dan teratur kedepannya.

DAFTAR PUSTAKA

https://lamanit.com/algoritma-greedy/#Pengertian_Algoritma_Greedy (Diakses pada 6 Maret 2024).

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf (Diakses pada 6 Maret 2024).

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf (Diakses pada 6 Maret 2024).

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Greedy-(2022)-Bag3.pdf (Diakses pada 6 Maret 2024).

LAMPIRAN

Link *repository*:

https://github.com/bastianhs/Tubes1_OogoeDiamonds

Link video youtube:

https://youtu.be/1jN8Y1oXtOQ?si=iB2bEn8MAoaR_as-