ALGORITMA GREEDY BY DENSITY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

Diajukan sebagai syarat menyelesaikan mata kuliah Strategi Algoritma (IF2211) Kelas RA
di Program Studi Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Sumatera

Oleh: Kelompok StockCat

Muhammad Fadil Ataullah Rifqi 122140205

Dosen Pengampu: Imam Ekowicaksono, S.Si., M.Si.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA

2025

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.

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 kalian implementasikan dengan algoritma *greedy* untuk bot kelompok kalian)

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.

- ***Game engine* :**

  https://github.com/haziqam/tubes1-IF2211-game-engine/releases/tag/v1.1.0

- ***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**

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*.

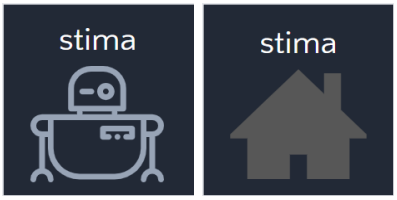
2. **Red Button/Diamond Button**

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.

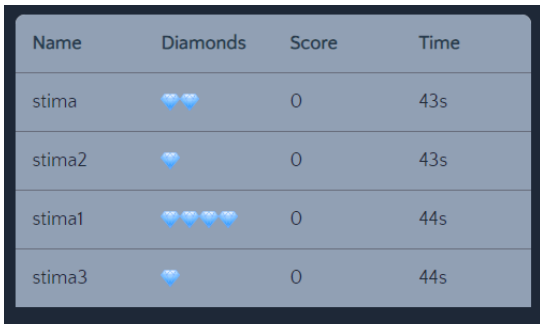
4. **Bots and Bases**



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

2.1 Dasar Teori

Algoritma Greedy adalah metode populer dan sederhana untuk memecahkan persoalan optimasi, yang bertujuan mencari solusi maksimal atau minimal, dengan berpegang pada prinsip "*take what you can get now!*". Algoritma ini membangun solusi langkah demi langkah, di mana pada setiap langkah dipilih keputusan terbaik lokal dengan harapan akan mencapai optimum global, tanpa bisa meninjau ulang keputusan sebelumnya. Meskipun melibatkan elemen seperti himpunan kandidat, fungsi seleksi, dan fungsi kelayakan, algoritma greedy tidak selalu menghasilkan solusi optimum global karena tidak mengeksplorasi semua kemungkinan dan sangat bergantung pada fungsi seleksi, sehingga terkadang hanya memberikan solusi suboptimal. Namun, ia berguna untuk solusi hampiran yang cepat pada masalah kompleks dan jika terbukti optimal, hal itu memerlukan pembuktian matematis, dengan contoh penerapan pada persoalan penukaran uang dan pemilihan aktivitas[1].

2.2 Cara Kerja Program

Cara kerja bot dimulai dengan pendaftaran pada board melalui HTTP request. Setelah terdaftar, bot akan secara berkala mengirimkan data arah geraknya ke backend game, juga melalui HTTP request. Untuk menentukan arah gerak ini, diperlukan sebuah algoritma. Meskipun template bot telah menyediakan algoritma Random, algoritma tersebut kurang optimal. Oleh karena itu, saya akan mengimplementasikan algoritma greedy untuk menentukan arah gerak bot yang lebih baik. Berikut adalah langkah-langkah untuk menjalankan bot dengan algoritma yang diimplementasikan sendiri:

1. Download / Clone source code game engine Etimo Diamonds 2
2. Jalankan game dengan npm build dan npm start pada repository game
3. Download / Clone source code template bot
4. Buat file baru pada game/logic/

5. Implementasikan algoritma untuk menentukan arah gerak bot
6. Import class algoritma pada main file di main.py dan daftarkan pada CONTROLLERS
7. Jalankan bot dengan python main.py --logic {class yang diimport} --email={email} --name={nama untuk ditampilkan di board} --password={password} --team {team} --board {nomor board} atau bisa menjalankan batch dengan menggunakan .sh atau .bat

BAB III

APLIKASI STRATEGI *GREEDY*

3.1 Proses *Mapping*

Elemen-elemen algoritma greedy:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah
2. Himpunan solusi, S : berisi kandidat yang sudah dipilih
3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi
4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.
5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)
6. Fungsi obyektif : memaksimumkan atau meminimumkan

No	Elemen	Definisi
1	Himpunan kandidat	Seluruh object yang ada di board
2	Himpunan solusi	Object yang paling menguntungkan untuk diambil
3	Fungsi solusi	Diamond di dalam *inventory* bot harus berjumlah 5
4	Fungsi seleksi	Memilih object berdasarkan p (jarak) dan w (nilai)
5	Fungsi kelayakan	Mengecek apakah object yang dipilih masih layak untuk diambil

6	Fungsi Objective	Mengubah nilai p dari suatu object, berdasarkan kondisi *inventory*

3.2 Eksplorasi Alternatif Solusi Greedy

3.2.1 Solusi Greedy by Nilai Diamond

Solusi *Greedy by Weight* berfokus pada pemilihan Diamond nilai tertinggi (weight) tanpa mempertimbangkan jarak Diamond terhadap posisi bot. Dalam pendekatan ini, bot akan memilih Diamond yang memiliki nilai paling besar, selama masih layak diambil (memenuhi fungsi kelayakan). Strategi ini cocok diterapkan dalam situasi di mana tujuan utama adalah mengumpulkan skor setinggi mungkin, dan tidak terdapat tekanan dari segi waktu atau jarak tempuh. Namun, kelemahan dari pendekatan ini adalah potensi inefisiensi karena bot bisa saja mengejar Diamond yang sangat jauh, yang secara keseluruhan justru merugikan karena waktu dari game terbatas. Oleh karena itu, meskipun greedy by weight efektif dalam memaksimalkan hasil lokal, ia belum tentu optimal secara global jika aspek lain seperti waktu dan kondisi medan.

3.2.2 Solusi Greedy by Jarak Diamond

Solusi *Greedy by Profit* menggunakan pendekatan yang berlawanan dengan greedy by weight, yaitu dengan hanya mempertimbangkan jarak terdekat dari Diamond terhadap posisi bot. Dalam strategi ini, bot akan selalu memilih Diamond yang paling dekat, terlepas dari seberapa besar nilai Diamond tersebut. Tujuan dari pendekatan ini adalah untuk memaksimalkan efisiensi pergerakan bot, baik dari segi waktu maupun energi. Pengabaian nilai Diamond ini merupakan kelemahan utama, karena bot bisa menghabiskan waktu untuk Diamond padahal ada jalan lain dengan keuntungan lebih besar yang terlewatkan.

3.2.3 Solusi Greedy by Density (Nilai Diamond / Jarak)

Solusi Greedy by Density menawarkan pendekatan yang lebih seimbang dalam pemilihan Diamond dengan menghitung rasio antara nilai Diamond dan jarak yang harus ditempuh bot untuk mencapainya, menggunakan formula **Density = Nilai Diamond / Jarak ke Diamond.** Pada setiap tahap pengambilan keputusan, bot akan mengevaluasi semua Diamond yang tersedia, memilih Diamond dengan nilai densitas tertinggi. Tujuan utama strategi ini adalah memaksimalkan perolehan nilai per unit jarak, sehingga secara

efektif mengoptimalkan efisiensi waktu dan energi dengan menyeimbangkan antara seberapa berharga sebuah Diamond dan seberapa sulit untuk mendapatkannya.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Pada sub-bab sebelumnya, telah dieksplorasi tiga alternatif solusi menggunakan algoritma greedy untuk menentukan arah gerak bot dalam mengumpulkan diamond: Greedy berdasarkan Nilai Diamond, Greedy berdasarkan Jarak Diamond, dan Greedy berdasarkan Densitas (Nilai Diamond / Jarak). Masing-masing pendekatan memiliki karakteristik efisiensi dan efektivitas yang berbeda.

Efisiensi dalam konteks ini mengacu pada penggunaan sumber daya seperti waktu atau energi (jumlah langkah) yang dikeluarkan bot untuk mencapai tujuan. **Efektivitas** mengacu pada seberapa baik bot mencapai tujuan utamanya, yaitu memaksimalkan total nilai diamond yang dikumpulkan.

1. **Solusi Greedy by Nilai Diamond**

    a. **Efektivitas Tinggi (Potensial):** Pendekatan ini berpotensi sangat efektif dalam mengumpulkan skor tinggi jika bot berhasil mendapatkan diamond bernilai besar. Fokus utamanya adalah memaksimalkan nilai per diamond yang diambil.

    b. **Efisiensi Rendah:** Kelemahan utama terletak pada efisiensi. Bot mungkin mengabaikan diamond bernilai lebih kecil yang sangat dekat dan malah mengejar diamond bernilai sangat tinggi yang lokasinya jauh. Hal ini dapat menyebabkan bot menghabiskan banyak waktu dan langkah hanya untuk satu diamond, yang mungkin tidak optimal jika waktu permainan terbatas atau ada banyak diamond kecil yang bisa dikumpulkan dengan cepat. Ini dapat mengakibatkan total skor yang lebih rendah dalam jangka panjang karena waktu terbuang untuk perjalanan jauh.

2. **Solusi Greedy by Jarak Diamond**

    a. **Efektivitas Rendah (Potensial):** Pendekatan ini cenderung kurang efektif dalam memaksimalkan skor total. Dengan selalu memilih diamond terdekat, bot bisa saja

terus-menerus mengambil diamond bernilai sangat kecil, sementara diamond bernilai besar yang mungkin tidak terlalu jauh menjadi terabaikan.

b. **Efisiensi Tinggi (Pergerakan):** Dari segi pergerakan, strategi ini sangat efisien. Bot akan meminimalkan jarak tempuh untuk setiap diamond yang diambil, sehingga dapat mengumpulkan banyak diamond dalam waktu singkat atau dengan energi minimal. Namun, efisiensi pergerakan ini tidak selalu sejalan dengan efektivitas pengumpulan skor.

3. **Solusi Greedy by Density (Nilai Diamond / Jarak)**

    a. **Efektivitas Seimbang/Tinggi:** Pendekatan ini menawarkan keseimbangan terbaik antara nilai diamond dan jarak. Dengan menghitung rasio nilai terhadap jarak, bot akan memprioritaskan diamond yang memberikan "imbalan" terbaik per langkah. Ini berarti bot tidak akan mengejar diamond yang sangat jauh meskipun nilainya tinggi jika ada diamond lain yang nilainya cukup baik dan jauh lebih dekat. Sebaliknya, bot juga tidak akan terjebak mengambil diamond bernilai sangat rendah hanya karena dekat jika ada opsi yang sedikit lebih jauh namun nilainya jauh lebih signifikan.

    b. **Efisiensi Seimbang/Tinggi:** Karena memperhitungkan jarak dalam formulanya, solusi ini cenderung lebih efisien daripada greedy by nilai murni. Bot tidak akan melakukan perjalanan yang tidak proporsional dengan nilai diamond yang akan didapat. Ini mengarah pada penggunaan waktu dan langkah yang lebih optimal secara keseluruhan untuk memaksimalkan skor.

Secara keseluruhan, solusi **Greedy by Density** muncul sebagai pendekatan yang paling seimbang dan berpotensi paling efektif untuk strategi pengumpulan diamond oleh bot. Algoritma ini berhasil memaksimalkan perolehan skor per unit jarak atau waktu dengan mempertimbangkan baik nilai diamond maupun jarak tempuh, menjadikannya strategi optimal dalam skenario permainan dengan sumber daya terbatas seperti waktu atau energi bot. Berbeda dengan **Greedy by Nilai Diamond** yang berisiko tinggi menjadi tidak efisien karena berpotensi mengejar diamond bernilai tinggi yang sangat jauh meskipun cocok jika tidak ada batasan waktu dan nilai

diamond sangat bervariasi dan **Greedy by Jarak Diamond** yang sangat efisien dalam pergerakan namun seringkali tidak efektif memaksimalkan skor karena mengabaikan nilai diamond solusi ini berguna jika tujuan utama adalah jumlah diamond terbanyak atau nilainya seragam. Dengan menghindari jebakan kedua metode tersebut, yaitu mengejar diamond jauh bernilai tinggi secara membabi buta atau hanya mengambil diamond dekat bernilai rendah, solusi **Greedy by Density** cenderung menjadi pilihan yang lebih robust dan adaptif untuk berbagai kondisi permainan dibandingkan dua alternatif lainnya.

3.4 Strategi Greedy yang Dipilih

Setelah melakukan eksplorasi terhadap beberapa alternatif solusi greedy dan menganalisis efisiensi serta efektivitas masing-masing, strategi yang akhirnya dipilih untuk diimplementasikan adalah **Solusi Greedy by Density (Nilai Diamond / Jarak)**.

Pemilihan strategi ini didasarkan pada kesimpulan analisis di sub-bab 3.3, yang menunjukkan bahwa pendekatan Greedy by Density menawarkan keseimbangan terbaik antara memaksimalkan perolehan nilai diamond dan efisiensi pergerakan bot. Dengan menghitung rasio antara nilai diamond dan jarak yang harus ditempuh, bot akan selalu memprioritaskan target yang memberikan "imbalan" tertinggi per unit usaha (jarak atau waktu).

Strategi ini dianggap lebih unggul dibandingkan dua alternatif lainnya karena:

1. Menghindari potensi inefisiensi dari **Greedy by Nilai Diamond**, di mana bot bisa menghabiskan terlalu banyak waktu atau langkah untuk mengejar diamond bernilai tinggi yang lokasinya sangat jauh, sehingga mengabaikan peluang lain yang lebih cepat dijangkau.
2. Mengatasi kelemahan **Greedy by Jarak Diamond**, yang meskipun efisien dalam pergerakan, seringkali kurang efektif dalam memaksimalkan skor total karena cenderung mengumpulkan diamond bernilai rendah hanya karena lokasinya dekat.

Dengan demikian, strategi Greedy by Density diharapkan dapat menghasilkan performa bot yang lebih optimal secara keseluruhan, karena menyeimbangkan antara seberapa berharga sebuah

diamond dan seberapa sulit atau jauh untuk mendapatkannya, sehingga cocok untuk kondisi permainan dengan waktu atau energi yang terbatas.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

```
CLASS StockCat EXTENDS BaseLogic:
      PROSEDURE init:
            self_goal = null
            self_teleporters = null
            self_target_after_teleport = null

      FUNCTION <Position> next_move:
      // Fungsi Solusi
      IF inventory IS EQUAL 5:
            self_goal = base
      ELSE
            // Definisikan Himpunan Kandidat
            diamonds = board.diamonds
            bots = board.bots
            teleports = board.teleports
            reset_button = board.reset_button

            candidates = diamonds + teleporters + reset_button +
teleports

            // Fungsi Seleksi
            best_candidate = null
            best_p = -1
            best_w = 100
            best_d = 100

            FOR candidate, distance IN candidates:
                  // Fungsi Kelayakan
                  IF candidate IS EQUAL bots:
                  CONTINUE
                  IF candidate IS EQUAL teleporter:
```

```
                CONTINUE
                IF candidate IS EQUAL diamond:
                // Fungsi Objektif
                IF diamond.value IS EQUAL 1:
                     weigth = 2
                ELSE:
                     weigth = 4

                IF diamond.value + inventory IS MORE THAN 5:
                     profit = 1000
                ELSE:
                     profit = distance
                ELSE IF candidate IS EQUAL reset_button:
                weigth = 1
                profit = distances

                density = profit / weigth
                IF density IS LESS THAN best_d:
                best_candidate = candidate
                best_p = profit
                best_w = weigth
                best_d = density
           END FOR
           // Himpunan Solusi
           IF best_candidate IS NOT NULL:
                self_goal = best_candidate
                IF best_candidate IS EQUAL teleporter:
                self_teleporters = teleporters
                self_target_after_teleport =
get_target_after_teleport(best_candidate)
     END IF

     delta_x, delta_y = get_direction(
          current_position, self_goal.position
     )

     return delta_x, delta_y
```

4.1.2 Penjelasan Alur Program

Alur program untuk algoritma StockCat dirancang untuk menentukan tindakan bot pada setiap iterasi berdasarkan kondisi saat ini dan evaluasi terhadap target-target potensial. Implementasi ini secara fundamental mengadopsi pendekatan algoritma greedy untuk pengambilan keputusan ketika inventory bot belum mencapai kapasitas maksimal.

1. Evaluasi Kondisi inventory Awal (Fungsi Solusi)

Langkah pertama dalam fungsi next_move adalah pemeriksaan status inventory bot. Memeriksa apakah isi di dalam *inventory* berjumlah 5. Apabila kondisi ini terpenuhi, yang mengindikasikan bahwa inventory bot telah terisi penuh, maka self_goal akan secara langsung ditetapkan ke base.

2. Mekanisme Pencarian dan Seleksi Target (Jika inventory Belum Penuh)

Jika kondisi kapasitas inventory maksimal belum tercapai (ELSE), bot akan menjalankan prosedur untuk mengidentifikasi dan memilih target berikutnya. Proses ini terdiri dari beberapa sub-tahapan:

a. **Pembentukan Himpunan Kandidat Target**: bot akan mengumpulkan informasi mengenai semua entitas yang relevan dari lingkungan permainan. Entitas-entitas ini meliputi diamonds (berlian), bots (bot lain), teleports (teleporter), dan reset_button (tombol reset). Selanjutnya, sebuah himpunan agregat bernama candidates dibentuk, yang berisi: diamonds + teleporters + reset_button + teleports.

b. **Proses Seleksi Kandidat (Inti Implementasi Algoritma Greedy)**: Sebelum iterasi dimulai, beberapa variabel diinisialisasi untuk melacak kandidat terbaik yang ditemukan: best_candidate disetel ke null, best_p, best_w, dan best_d. Nilai inisialisasi best_d yang tinggi memastikan bahwa kandidat valid pertama yang dievaluasi akan dianggap sebagai yang terbaik sementara. Selanjutnya, dilakukan iterasi kandidat, untuk setiap entitas dalam himpunan candidates, beserta distance dari posisi bot saat ini ke kandidat tersebut.

c. **Validasi Kelayakan Kandidat (Fungsi Kelayakan)**: Filter diterapkan untuk menentukan apakah sebuah kandidat akan dipertimbangkan lebih lanjut Kandidat yang merupakan bots akan diabaikan. Kandidat yang merupakan teleporter juga akan diabaikan dari proses perhitungan fungsi objektif. Berdasarkan filter ini, hanya kandidat berjenis diamond dan reset_button yang akan dilibatkan dalam perhitungan densitas.

d. **Kalkulasi Nilai Objektif (Fungsi Objektif) untuk Kandidat yang Valid**: Untuk Weight kandidat diamond akan bernilai 2 jika diamond.value adalah 1, dan 4 untuk value 2. profit akan bernilai 1000 (mengubah jarak menjadi sebesar mungkin) jika pengambilan berlian tersebut akan menyebabkan total item dalam inventory melebihi 5. Jika tidak, profit akan sama dengan distance ke berlian. Untuk weigth reset_button ditetapkan sebagai 1. profit ditetapkan sebagai distances (diasumsikan sebagai distance ke tombol reset).

e. **Keputusan Lokal Berbasis Greedy**: Nilai density dihitung sebagai profit / weight. Jika density dari kandidat yang sedang dievaluasi lebih rendah daripada best_d (densitas terbaik yang tersimpan), maka kandidat tersebut akan menjadi best_candidate baru. Nilai best_p, best_w, dan best_d juga akan diperbarui.

f. **Penetapan Target Final (Himpunan Solusi)**: Jika sebuah best_candidate telah berhasil diidentifikasi melalui proses seleksi. Tujuan akhir bot ditetapkan ke posisi best_candidate ini.

3. Perhitungan Vektor Pergerakan

Setelah self_goal ditetapkan (baik ke base atau ke best_candidate yang dipilih), fungsi get_direction(current_position, self_goal.position) akan dipanggil. Fungsi ini adalah fungsi bawaan template yang bertugas menghitung delta_x dan delta_y, yang merupakan komponen vektor pergerakan dari posisi bot saat ini (current_position) menuju posisi target (self_goal.position).

4.2 Struktur Data yang Digunakan

Pada implementasi logika bot StockCat, terdapat beberapa jenis struktur data yang digunakan untuk mengatur dan mengelola informasi dari objek-objek permainan. Pemilihan struktur data tersebut disesuaikan dengan kebutuhan algoritmik dan efisiensi dalam pengambilan keputusan. Berikut ini adalah uraian struktur data yang digunakan:

a. List (Daftar)

Tipe data list digunakan untuk menyimpan sekumpulan objek permainan, seperti berlian, teleporter, dan tombol reset. List dipilih karena mendukung proses iterasi secara sekuensial yang dibutuhkan dalam proses pencarian kandidat terbaik. Contoh penggunaannya meliputi:

Menyimpan seluruh objek diamond:

```
diamonds = [d for d in board.game_objects if d.type == "DiamondGameObject"]
```

Menyimpan pasangan teleporter:

```
self.teleporters[pair_id] = [tele1, tele2]
```

Struktur list ini memungkinkan proses seleksi berdasarkan jarak atau rasio nilai (density) dalam kompleksitas waktu linear (O(n)), yang efisien dalam konteks permainan real-time dengan jumlah objek terbatas.

b. Dictionary (Kamus)

Tipe data dictionary digunakan untuk mengelompokkan pasangan teleporter berdasarkan pair_id. Setiap pasangan disimpan dengan key berupa ID unik dan value berupa list dua teleporter. Dictionary memberikan kecepatan akses konstan O(1) dalam mengambil data berdasarkan kunci.
Contoh implementasinya adalah:

```
self.teleporters: Dict[str, List[GameObject]]
```

Penggunaan dictionary sangat efektif untuk memetakan hubungan satu-ke-satu atau satu-ke-banyak, seperti dalam kasus pasangan teleportasi yang perlu diakses dengan cepat.

c. Tuple

Tuple digunakan untuk menyimpan pasangan nilai tetap, seperti koordinat pergerakan bot dan pasangan objek dengan nilai evaluasi tertentu (misalnya jarak atau densitas). Karena bersifat immutable, tuple aman digunakan dalam proses komputasi yang tidak membutuhkan modifikasi data.

Contoh penggunaan tuple:

Sebagai hasil pergerakan bot:

```
def next_move(...) -> Tuple[int, int]
```

Sebagai pasangan kandidat dan jaraknya:

```
candidates.append((diamond, distance))
```

d. Optional

Struktur data Optional digunakan pada variabel yang nilainya bisa berubah atau tidak tersedia pada waktu tertentu. Hal ini penting untuk mencegah kesalahan akses (misalnya NoneType error) ketika variabel belum diinisialisasi. Penggunaannya terlihat pada:

```
self.goal_position: Optional[Position]
self.target_after_teleport: Optional[Position]
```

Dengan menggunakan Optional, kode menjadi lebih aman dan stabil karena memaksa adanya pengecekan kondisi sebelum penggunaan data.

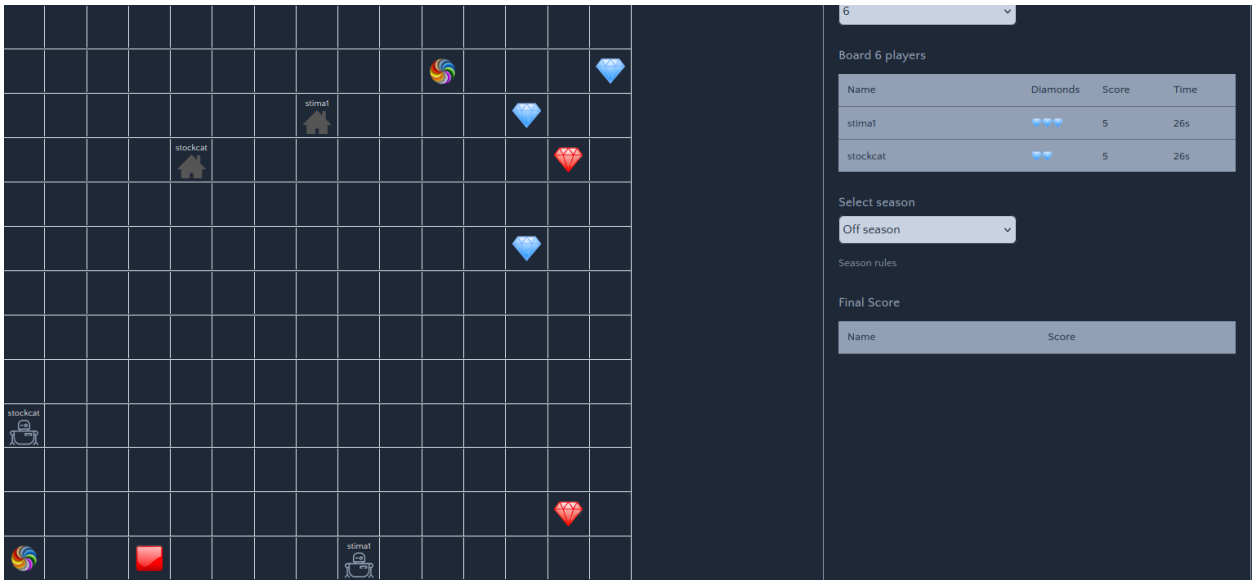
4.3 Pengujian Program

1. Skenario Pengujian

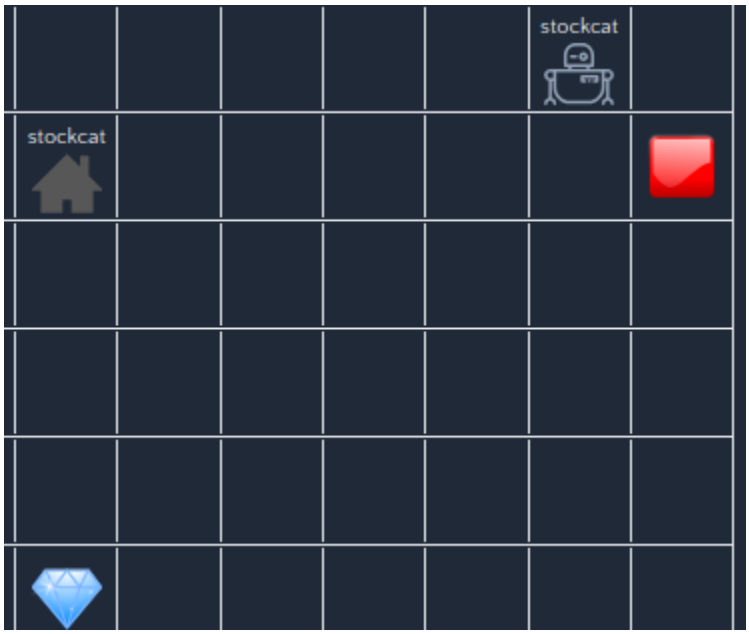
Skenario 1:

Skenario 2:

Skenario 3:



Skenario 4:



2. Hasil Pengujian dan Analisis

Skenario 1:

D1 = diamond atas, D2 = diamond bawah, D1:w=2;p=3;d=1.5,D2:w=2;p=4;d=2

Pada kasus ini bot akan memilih diamond yang atas karena d/densitas Diamond atas lebih kecil dibanding diamond 2. Kenapa memilih densitas yang lebih kecil? Karena p disini adalah jarak yang dibutuhkan untuk mencapai diamond semakin kecil p maka semakin untung karena d = p/w maka d ≡ p.

Skenario 2:

D1 = diamond merah, D2 = diamond biru atas

D1:w=4;p=3;d=0.75,D2:w=2;p=1;d=0.5

Pada kasus ini bot akan memilih diamond biru karena d/densitas Diamond atas lebih kecil dibanding Diamond Merah meskipun nilai Diamond merah lebih besar daripada Diamond biru.

Skenario 3:

D1 = diamond merah, D2 = diamond biru atas

D1:w=4;p=9;d=2.25,D2:w=2;p=7;d=3.5

Pada kasus ini bot akan memilih diamond merah karena d/densitas Diamond atas lebih kecil dibanding Diamond biru meskipun nilai Diamond biru mempunyai jarak yang lebih dekat.

Skenario 4:

R1 = reset btton, D2 = diamond biru atas, R1:w=1;p=2;d=2,D2:w=2;p=10;d=5

Pada kasus ini bot akan memilih Reset Button karena d/densitas Reset Button lebih kecil

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan implementasi dan pengujian yang telah dilakukan pada bot StockCat dalam permainan Diamonds, dapat disimpulkan bahwa:

1. **Strategi Greedy by Density** terbukti sebagai pendekatan yang paling efektif dan seimbang dibandingkan dua alternatif lainnya (Greedy by Nilai dan Greedy by Jarak). Pendekatan ini menggabungkan aspek nilai objek dan jarak tempuh dalam satu metrik rasional, yaitu:

   $$\text{Density} = \frac{p}{w}$$

   yang berarti semakin kecil densitas, semakin menguntungkan sebuah objek untuk diambil.
2. Bot secara konsisten mampu memilih target (diamond atau reset button) yang memberikan imbal hasil terbaik per satuan jarak. Hal ini mengoptimalkan performa bot dalam kondisi permainan yang dinamis, terbatas waktu, dan penuh kompetisi.
3. Hasil pengujian terhadap berbagai skenario memperlihatkan bahwa keputusan bot selaras dengan strategi yang diharapkan. Bot mampu:
   a. Memprioritaskan diamond dekat bila nilainya sebanding,
   b. Mengorbankan jarak bila nilai objek sangat besar,
   c. Mengabaikan objek yang secara efisien tidak menguntungkan,
   d. Beradaptasi memilih reset button jika semua opsi diamond terlalu mahal secara densitas.
4. Pemanfaatan struktur data seperti List, Dictionary, Tuple, dan Optional sangat berperan dalam mendukung efisiensi akses dan evaluasi objek pada papan permainan, memastikan logika bot dapat berjalan secara stabil dan optimal.
5. Secara keseluruhan, implementasi bot StockCat berhasil menunjukkan performa yang optimal dalam konteks strategi algoritma greedy, dengan keputusan yang terukur dan efisien berdasarkan kondisi papan permainan secara real-time.

5.2 Saran

Adapun beberapa saran untuk pengembangan lebih lanjut dari bot StockCat adalah sebagai berikut:

1. **Adaptasi terhadap Lawan (Anti-Tackle)**

   Saat ini bot belum secara aktif menghindari bot lawan. Pengembangan strategi berbasis *game theory* atau *path prediction* untuk menghindari tackle atau bahkan menyerang balik bisa meningkatkan performa dalam pertandingan kompetitif.

2. **Pembobotan Dinamis Berdasarkan Waktu**

   Densitas saat ini dihitung statis. Bot dapat ditingkatkan dengan *dynamic weighting*, misalnya memperbesar bobot jarak seiring berkurangnya waktu permainan, agar lebih agresif dalam menyimpan poin.

LAMPIRAN

A. **Repository Github (https://github.com/FadilRifqi/Tubes1_StockCat)**

B. **Video Penjelasan (https://youtu.be/ab1V8eOnRPU?si=OreftospejeGqKjv)**

DAFTAR PUSTAKA

[1] R. Munir, "Algoritma Greedy (Bagian 1)," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB, 2021.