

LAPORAN TUGAS BESAR IF2211 STRATEGI ALGORITMA

**Penggunaan Algoritma Greedy
dalam Optimalisasi Bot Permainan Diamonds**



Dosen Pengampu : Winda Yulita, M.Cs.

Asisten Pembimbing: Mohamad Meazza Aprilianda

Disusun Oleh:

Kelompok 15 – Placeholder

Adi Septriansyah (123140021)

Alfino Pardiansyah Hutahaean (122140126)

Ariq Ramadhinov Ronny (123140105)

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

DAFTAR ISI	2
BAB I	
DESKRIPSI TUGAS	1
1.1 Abstraksi	1
BAB II	
LANDASAN TEORI	5
2.1 Dasar Teori	5
2.2 Cara Kerja Permainan Diamonds	5
2.3 Cara Mengimplementasikan Algoritma Greedy Pada Bot Permainan Diamonds	6
BAB III	
APLIKASI STRATEGI GREEDY	7
3.1 Elemen Elemen Algoritma Greedy	7
3.2 Konsep Heuristik Untuk Eksplorasi Alternatif Solusi Greedy	8
3.3 Analisis Efisiensi dan Efektivitas dari Alternatif Solusi Greedy	9
3.4 Strategi Greedy yang Dipilih	10
BAB IV	
IMPLEMENTASI DAN PENGUJIAN	12
4.1 Implementasi Algoritma Greedy dalam Bot Diamonds Menggunakan Python	12
4.2 Implementasi Algoritma Greedy dalam Bot Diamonds Menggunakan Pseudocode	15
4.3 Penjelasan Struktur Data dalam Program Bot Diamonds	19
4.4 Analisis Desain Solusi Algoritma	20
4.5 Kasus Pengujian	22
BAB V	
KESIMPULAN DAN SARAN	24
5.1 Kesimpulan	24
5.2 Saran	24
Lampiran	25
Daftar Pustaka	26

BAB I

DESKRIPSI TUGAS

1.1 Abstraksi

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.



Gambar 1. 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 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>

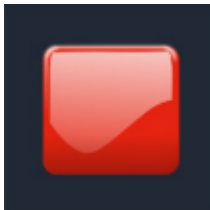
Komponen-komponen dari permainan Diamonds antara lain:

1. Diamonds



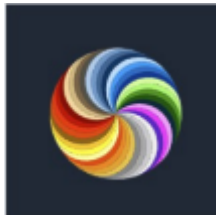
Untuk memenangkan pertandingan, kita harus mengumpulkan diamond ini sebanyak-banyaknya dengan melewati/melangkahinya. Terdapat 2 jenis diamond yaitu diamond biru dan diamond merah. Diamond merah bernilai 2 poin, sedangkan yang biru bernilai 1 poin. Diamond akan di-regenerate secara berkala dan rasio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

2. Red Button/Diamond Button



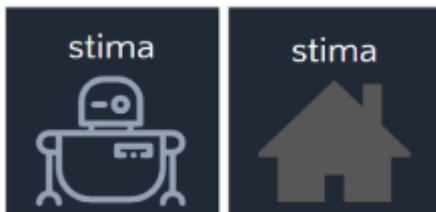
Ketika red button ini dilewati/dilangkahi, semua diamond (termasuk red diamond) akan di-generate kembali pada board dengan posisi acak. Posisi red button ini juga akan berubah secara acak jika red button ini dilangkahi.

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

Name	Diamonds	Score	Time
stima	♥♥	0	43s
stima2	♥	0	43s
stima1	♥♥♥♥	0	44s
stima3	♥	0	44s

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 menempa 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.

Panduan Penggunaan

Adapun panduan mengenai cara instalasi, menjalankan permainan, membuat bot, melihat visualizer/frontend, dan mengatur konfigurasi permainan dapat dilihat melalui tautan berikut. <https://docs.google.com/document/d/1L92Axb89yIkom0b24D350Z1QAr8rujvHof7-kXRAp7c>

Mekanisme Teknis Permainan Diamonds

Permainan ini merupakan permainan berbasis web, sehingga setiap aksi yang dilakukan – mulai dari mendaftarkan bot hingga menjalankan aksi bot – akan memerlukan HTTP request terhadap API endpoint tertentu yang disediakan oleh backend. Berikut adalah urutan requests yang terjadi dari awal mula permainan.

1. Program bot akan mengecek apakah bot sudah terdaftar atau belum, dengan mengirimkan POST request terhadap endpoint `/api/bots/recover` dengan body berisi email dan password bot. Jika bot sudah terdaftar, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut. Jika tidak, backend akan memberikan response code 404.
2. Jika bot belum terdaftar, maka program bot akan mengirimkan POST request terhadap endpoint `/api/bots` dengan body berisi email, name, password, dan team. Jika berhasil, maka backend akan memberikan response code 200 dengan body berisi id dari bot tersebut.
3. Ketika id bot sudah diketahui, bot dapat bergabung ke board dengan mengirimkan POST request terhadap endpoint `/api/bots/{id}/join` dengan body berisi board id yang diinginkan (`preferredBoardId`). Apabila bot berhasil bergabung, maka backend akan memberikan response code 200 dengan body berisi informasi dari board.
4. Program bot akan mengkalkulasikan move selanjutnya secara berkala berdasarkan kondisi board yang diketahui, dan mengirimkan POST request terhadap endpoint `/api/bots/{id}/move` dengan body berisi direction yang akan ditempuh selanjutnya (“NORTH”, “SOUTH”, “EAST”, atau “WEST”). Apabila berhasil, maka backend akan memberikan response code 200 dengan body berisi kondisi board setelah move tersebut. Langkah ini dilakukan terus-menerus hingga waktu bot habis. Jika waktu bot habis, bot secara otomatis akan dikeluarkan dari board.
5. Program frontend secara periodik juga akan mengirimkan GET request terhadap endpoint `/api/boards/{id}` untuk mendapatkan kondisi board terbaru, sehingga tampilan board pada frontend akan selalu ter-update.

BAB II

LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy merupakan salah satu metode yang paling dikenal dan sering digunakan untuk menyelesaikan persoalan optimasi. Persoalan optimasi sendiri merujuk pada masalah yang tujuannya adalah menemukan solusi terbaik (optimal), baik itu dalam bentuk maksimasi (mencari nilai terbesar) maupun minimasi (mencari nilai terkecil). Kunci dari algoritma Greedy terletak pada pendekatannya yang membentuk solusi secara bertahap, langkah demi langkah. Pada setiap langkah tersebut, algoritma akan membuat pilihan yang dianggap paling baik berdasarkan informasi yang tersedia pada saat itu, dengan harapan bahwa serangkaian pilihan ini akan mengarah pada solusi optimal.

Meskipun populer karena kesederhanaannya, penting untuk dipahami bahwa algoritma Greedy tidak selalu menjamin pencapaian solusi optimal. Dalam beberapa kasus, solusi yang dihasilkan mungkin bersifat sub-optimal atau pseudo-optimal. Hal ini disebabkan karena algoritma Greedy membuat keputusan tanpa mempertimbangkan konsekuensi jangka panjang dari pilihan yang dibuat pada setiap langkahnya. Algoritma greedy tidak melihat secara menyeluruh terhadap semua kemungkinan solusi yang ada. Keberhasilan algoritma Greedy sangat bergantung pada struktur masalah dan, yang lebih penting, pada bagaimana pilihan terbaik didefinisikan melalui fungsi seleksi.

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)
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

2.2 Cara Kerja Permainan Diamonds

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. Obyektif 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 tidak bertemu dengan bot lawan. Jika bot A menempa 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).
6. Selain itu, terdapat beberapa fitur tambahan seperti teleporter dan red button yang dapat digunakan apabila bot menuju posisi objek tersebut.
7. Apabila waktu seluruh bot telah berakhir, maka permainan berakhir. Score masing-masing pemain akan ditampilkan pada tabel Final Score di sisi kanan layar

2.3 Cara Mengimplementasikan Algoritma Greedy Pada Bot Permainan Diamonds

Sesuai dengan folder bot engine yang dapat diunduh melalui tautan yang terdapat pada Bab 1, pengembangan bot engine dapat dilakukan melalui file `./game/logic/random.py`.

Pengembangan bot dengan algoritma greedy memiliki beberapa keterbatasan. Misalnya, kita tidak diizinkan untuk mengubah struktur program atau kode program selain file yang berisi program bot di folder `tubes1-IF2211-bot-starter-pack-1.0.1\game\logic`. Saat mengembangkan bot dalam folder `./game/logic`, kita harus memastikan bahwa fungsi next move dalam class yang kita rancang memiliki dua parameter, yaitu variabel dengan class `'GameObject'` dan juga `'Board'`. Fungsi next move yang telah dibuat juga harus mengembalikan atau return dua value yaitu `delta_x` dan `delta_y` yang memiliki variasi nilai yaitu -1, 0, dan 1.

Untuk menjalankan satu bot, misalnya, satu bot yang menggunakan file `game/logic/random.py`, kita dapat menjalankan perintah `python main.py --logic Random --email=your_email@example.com --name=your_name --password=your_password --team etimo` dari root directory program bot engine. Selain itu, kita dapat menjalankan beberapa bot sekaligus dengan menjalankan perintah `./runbots.bat` untuk windows.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Elemen Elemen Algoritma Greedy

Berdasarkan cara bermain dan kondisi permainan, dapat diketahui bahwa bot akan melangkah dengan kecepatan 1 langkah/detik dan juga terdapat 2 jenis diamond, yakni red diamond yang berbobot 2 poin dan blue diamond yang berbobot 1 poin. Artinya, objek yang menjadi properti beban adalah langkah/waktu (kecepatan) sedangkan objek yang menjadi properti keuntungan adalah diamond.

Bot yang digunakan dalam laporan ini mengimplementasikan gabungan konsep algoritma *Greedy by Cost* (jarak) dan *Greedy by Profit* (poin) yaitu mencari objek objek diamond yang memiliki poin terbanyak secara kumulatif lalu menghitung jarak berdasarkan pengurangan dari poin kumulatif diamond dan objek sekitarnya.

- Himpunan Kandidat :
Kandidat dalam algoritma ini adalah objek diamond yang berada di papan permainan. Ini adalah objek yang coba dikumpulkan oleh bot, yang tersebar di seluruh papan.
- Himpunan Solusi :
Terdiri dari kemungkinan gerakan yang dapat dilakukan oleh bot untuk mengumpulkan diamond dan mungkin kembali ke base-nya. Ini mencakup bergerak ke diamond terdekat atau menggunakan teleporter jika dapat mengurangi jarak ke diamond target.
- Fungsi Solusi :
Mengevaluasi gerakan terbaik berdasarkan berbagai kriteria seperti jarak ke diamond target, bobot diamond (apakah diamond tersebut merah atau tidak), nilai cluster (nilai diamond di sekitar), dan waktu yang tersisa sebelum permainan berakhir.
- Fungsi seleksi
Memilih gerakan selanjutnya untuk bot berdasarkan kombinasi faktor, seperti nilai heuristik (yang mengevaluasi jalur terbaik menuju diamond) dan jarak ke base (jika bot perlu kembali).
- Fungsi kelayakan :
Memeriksa apakah sebuah gerakan atau strategi dapat dilaksanakan. Dalam hal ini, bot menentukan apakah bot masih dapat melanjutkan untuk mengumpulkan diamond atau apakah bot perlu kembali ke base. Terdapat fungsi untuk memeriksa kondisi kapan bot harus kembali ke base, seperti ketika bot telah mengumpulkan cukup diamond atau ketika waktu yang tersisa tidak mencukupi untuk mencapai diamond.
- Fungsi objektif :
Mengevaluasi kelayakan atau desirabilitas dari setiap gerakan. Bot menghitung cost/biaya total untuk bergerak menuju sebuah diamond tertentu, dengan mempertimbangkan jarak, bobot, nilai cluster, dan penalti waktu.

3.2 Konsep Heuristik Untuk Eksplorasi Alternatif Solusi Greedy

Heuristik adalah fungsi atau metode estimasi yang digunakan untuk menilai seberapa dekat algoritma terhadap tujuan yang diinginkan. Dalam ilmu komputer, khususnya dalam pencarian jalur dan kecerdasan buatan, heuristik digunakan untuk mempercepat proses pencarian dengan memberikan petunjuk tentang arah yang harus diambil. Heuristik tidak menjamin solusi optimal, tetapi sering kali cukup efektif dalam menghasilkan solusi yang baik dalam waktu yang lebih singkat. Heuristik yang baik akan memberikan estimasi yang mendekati nilai sebenarnya dan membantu algoritma dalam memilih langkah yang efisien.

1. Greedy by Distance (Jarak)
Bot memilih diamond yang terdekat berdasarkan jarak Manhattan. Fungsi ini digunakan dalam heuristik, di mana bot mencari diamond yang paling dekat untuk diambil.
2. Greedy by Weight (Bobot)
Bobot diamond diperhitungkan dalam strategi greedy ini. Diamond merah (yang bernilai 2 poin) memiliki bobot lebih tinggi daripada diamond biru (yang bernilai 1 poin). Bot juga mempertimbangkan bobot cluster diamond yang terdekat, yang dapat memengaruhi keputusan bot.
3. Greedy by Cluster (Kumpulan Diamond)
Bot juga mempertimbangkan cluster diamond, yaitu kumpulan diamond yang saling berdekatan dalam radius tertentu. Strategi ini menghitung total poin diamond di sekitar posisi bot. Dengan pendekatan ini, bot akan memilih diamond yang ada dalam cluster dengan total poin tertinggi.
4. Greedy by Time (Waktu Tersisa)
Waktu yang tersisa juga diperhitungkan dalam pemilihan gerakan bot. Bot akan memilih diamond berdasarkan heuristik yang mempertimbangkan waktu yang tersisa. Bot menambahkan penalti waktu (time penalty) ke dalam perhitungan total biaya untuk setiap langkah, yang memotivasi bot untuk bermain disekitar base jika waktu sudah terbatas.
5. Greedy by Return to Base (Kembali ke Base)
Greedy Return to base yaitu strategi greedy yang memprioritaskan bot untuk kembali ke base jika inventory sudah penuh atau waktu pada *Greedy by Time* tidak memungkinkan untuk mengambil diamond lebih banyak. Bot akan kembali ke base jika sudah memiliki 3 diamond atau lebih, atau jika waktu untuk bergerak ke diamond berikutnya lebih besar daripada waktu yang tersisa.
6. Greedy by Teleporter (Menggunakan Teleporter)
Jika ada teleporter yang bisa mengurangi jarak perjalanan, bot akan memilih untuk menggunakan teleporter.

3.3 Analisis Efisiensi dan Efektivitas dari Alternatif Solusi Greedy

Pada bagian ini, dilakukan analisis efisiensi dan efektivitas dari berbagai solusi Greedy yang mungkin diterapkan dalam permainan Diamonds. Setiap solusi akan dievaluasi berdasarkan kemampuan bot untuk mengumpulkan poin sebanyak mungkin dan efisiensi waktu dalam mencapai tujuan.

- Greedy by Distance (Jarak)

Dalam strategi Greedy by Distance, bot memilih untuk bergerak menuju diamond yang terdekat berdasarkan jarak Manhattan. Dengan pendekatan ini, bot selalu memilih langkah yang mengarah ke diamond dengan jarak terdekat tanpa memperhatikan nilai atau jenis diamond tersebut.

Strategi ini sangat efisien dalam hal waktu perjalanan karena bot selalu bergerak ke diamond terdekat. Namun, strategi ini dapat menjadi tidak efisien jika bot melewati diamond yang lebih bernilai hanya karena jaraknya yang lebih jauh. Meskipun demikian, dalam situasi di mana waktu sangat penting, pendekatan ini meminimalkan waktu yang terbuang untuk perjalanan.

Dalam hal efektivitas, strategi ini kurang optimal karena bot bisa saja mengumpulkan diamond biru yang bernilai lebih rendah, meskipun ada kesempatan untuk mendapatkan lebih banyak poin dari diamond merah yang lebih jauh. Sehingga, meskipun efisien, bot mungkin tidak akan mengumpulkan jumlah poin maksimal.

- Greedy by Weight (Bobot Diamond)

Strategi Greedy by Weight mengutamakan pemilihan diamond dengan nilai tertinggi terlebih dahulu. Diamond merah, yang bernilai 2 poin, akan diprioritaskan oleh bot, sedangkan diamond biru dengan nilai 1 poin akan diambil jika tidak ada pilihan lain.

Meskipun lebih fokus pada poin daripada jarak, strategi ini bisa kurang efisien dalam hal waktu. Jika diamond merah terletak jauh dari bot, bot akan menghabiskan banyak waktu untuk mencapainya, yang mengurangi kecepatan pergerakan bot secara keseluruhan. Oleh karena itu, bot bisa mengabaikan kesempatan untuk mengumpulkan diamond biru yang lebih dekat.

Strategi ini cukup efektif dalam memaksimalkan poin yang diperoleh, tetapi kurang efektif dalam mengoptimalkan waktu karena bot mungkin akan menghabiskan banyak waktu hanya untuk mencapai diamond merah yang terletak jauh.

- Greedy by Cluster (Kumpulan Diamond)

Strategi Greedy by Cluster mempertimbangkan kelompok diamond yang terletak berdekatan dalam radius tertentu. Bot akan memilih diamond yang ada dalam cluster dengan total poin tertinggi, yang memungkinkan bot untuk mengumpulkan banyak diamond sekaligus dalam satu perjalanan.

Strategi ini cukup efisien dalam hal waktu ketika diamond terkelompok dalam satu area. Bot bisa mengumpulkan beberapa diamond sekaligus tanpa harus kembali bolak-balik. Namun, jika diamond tersebar merata, strategi ini akan lebih sulit diterapkan dan tidak efisien.

Strategi ini sangat efektif pada papan permainan dengan diamond yang memiliki banyak cluster. Bot akan mengumpulkan banyak diamond sekaligus, namun jika diamond tersebar luas, bot mungkin akan kehilangan waktu yang lebih banyak untuk mencari cluster yang menguntungkan.

- Greedy by Time (Waktu Tersisa)

Dalam strategi Greedy by Time, bot memilih diamond berdasarkan waktu yang tersisa. Jika waktu terbatas, bot akan memilih untuk kembali ke base dan menyimpan diamond yang telah dikumpulkan, memastikan tidak ada waktu yang terbuang.

Strategi ini sangat efisien dalam hal mengelola waktu, memastikan bot tidak kehabisan waktu sebelum bisa kembali ke base. Namun, bot bisa kehilangan kesempatan untuk mengumpulkan

diamond lebih banyak jika terlalu banyak waktu dihabiskan untuk kembali ke base lebih cepat.

Efektif jika waktu sudah terbatas, tetapi tidak efektif dalam memaksimalkan poin jika masih ada waktu yang cukup untuk mengumpulkan lebih banyak diamond.

- Greedy by Return to Base (Kembali ke Base)

Strategi Greedy by Return to Base mengarahkan bot untuk kembali ke base jika inventory sudah penuh atau jika waktu tersisa sudah tidak memungkinkan untuk mengumpulkan lebih banyak diamond.

Strategi ini cukup efisien dalam menghindari pemborosan waktu karena bot tidak akan terus mengumpulkan diamond setelah inventory penuh. Namun, efisiensinya tergantung pada kecepatan bot dalam mengumpulkan diamond sebelum kembali ke base.

Strategi ini lebih efektif dalam menghindari kerugian di akhir permainan, tetapi tidak akan memaksimalkan poin yang dikumpulkan.

- Greedy by Teleporter (Menggunakan Teleporter)

Bot menggunakan teleporters untuk menghemat waktu perjalanan, berpindah ke lokasi yang lebih jauh dengan lebih cepat.

Teleporter dapat meningkatkan efisiensi waktu secara signifikan, karena bot bisa berpindah jauh lebih cepat daripada harus menempuh jalan biasa. Namun, penggunaan teleport yang berlebihan dapat menyebabkan waktu terbuang jika teleport berulang kali digunakan tanpa alasan yang jelas.

Efektif untuk mempercepat perjalanan, namun harus digunakan dengan hati-hati agar tidak mengarah pada penggunaan waktu yang tidak efisien.

3.4 Strategi Greedy yang Dipilih

Logika bot dapat dilihat sebagai pendekatan gabungan dari berbagai solusi greedy. Bot mencoba mengambil aspek poin terbesar (melalui weight) dan jarak terdekat, namun dimodifikasi oleh serangkaian heuristik tambahan seperti perlakuan khusus untuk diamond berdekatan, penggunaan teleporter, dan kondisi-kondisi spesifik untuk kembali ke base atau mengambil red button. Ini merupakan upaya untuk mengatasi kelemahan yang sering ditemukan pada strategi Greedy yang terlalu sederhana. Sebagai contoh, strategi jarak terdekat murni (Greedy by Distance) mungkin sangat efisien dalam hal langkah tetapi bisa menghasilkan skor rendah karena mengabaikan diamond bernilai tinggi yang sedikit lebih jauh. Sebaliknya, poin terbesar murni (Greedy by Weight) bisa sangat tidak efisien dalam pergerakan. Logika bot berusaha mencari keseimbangan dari berbagai variabel, dengan formula "Jarak - Weight" dan aturan-aturan tambahannya.

Perhitungan heuristik diperkirakan sebagai berikut

$$H = D - W + T$$

Di mana:

- H adalah nilai heuristik yang ingin dihitung.
- D adalah jarak terbaik antara posisi bot dan posisi diamond
- W adalah bobot diamond di posisi tujuan.
- T adalah penalti waktu yang dihitung berdasarkan waktu yang tersisa.

1. Jarak (Distance)

Menghitung jarak antara posisi bot dan diamond, serta mempertimbangkan kemungkinan penggunaan teleporter.

$$D = |x1 - x2| + |y1 - y2|$$

Di mana:

(x1,y1) adalah posisi bot.

(x2,y2) adalah posisi diamond.

Jika menggunakan teleporter, jarak ini akan ditambah dengan jarak dari bot ke teleport, ditambah jarak dari teleport ke diamond.

2. Beban Poin (Weight)

Weight dihitung dengan mempertimbangkan jumlah poin diamond terdekat, lalu memeriksa objek diamond lain yang ada disekitarnya. Diamond merah = 2 poin, diamond biru = 1 poin, tombol merah = 0/4 poin. Untuk tombol merah poin awalnya adalah 0. Namun, bisa berubah menjadi 4 poin jika jumlah poin kumulatif sekitar base sedikit.

$$W = Initial\ Diamond + (Cluster \times 0.5)$$

Di mana:

Initial diamond adalah poin awal dari diamond terdekat

Cluster adalah jumlah poin kumulatif dari diamond disekitar diamond terdekat(initial diamond)

3. Penalti Waktu (Time)

Penalti waktu dihitung berdasarkan sisa waktu yang tersedia. Semakin sedikit waktu yang tersisa, semakin besar penalti waktu untuk mengambil diamond tersebut. Rumusnya adalah:

$$T = D \times 0,5 \times \frac{1000ms}{Sisa\ waktu}$$

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy dalam Bot Diamonds Menggunakan Python

```
import random
from typing import Optional, List, Tuple

from game.logic.base import BaseLogic
from game.models import GameObject, Board, Position
from ..util import get_direction

class greedy12(BaseLogic):
    def __init__(self):
        super().__init__()
        self.directions = [(1, 0), (0, 1), (-1, 0), (0, -1)]
        self.inv_full: int = 5
        self.cluster_radius: int = 2
        self.base_time_penalty: float = 0.5

    def get_weight(self, obj: GameObject, board: Board, ignore_red=False) -> int:
        if obj.type == "DiamondGameObject":
            points = getattr(obj.properties, "points", 1)
            if ignore_red and points == 2:
                return 0
            return 2 if points == 2 else 1
        elif obj.type == "DiamondButtonGameObject":
            base = next((bot.properties.base for bot in board.bots
                        if hasattr(bot.properties, "base")), None)
            if base:
                nearby = [
                    o for o in board.game_objects
                    if o.type == "DiamondGameObject" and
                    abs(o.position.x - base.x) + abs(o.position.y - base.y) <= 4
                ]
                total_weight = sum(self.get_weight(o, board) for o in nearby)
                return 3 if total_weight <= 2 else 0
            return 0
        return 0

    def get_cluster_value(self, pos: Position, board: Board, ignore_red: bool) -> float:
        # Menghitung nilai cluster berdasarkan total poin diamond di sekitar
        cluster = [
            o for o in board.game_objects
            if o.type == "DiamondGameObject" and
            abs(o.position.x - pos.x) <= self.cluster_radius and
            abs(o.position.y - pos.y) <= self.cluster_radius
        ]
        return sum(self.get_weight(o, board, ignore_red) for o in cluster)
```

```

def get_best_path(self, pos1: Position, pos2: Position, board: Board) -> tuple[int,
Optional[GameObject]]:
    #distance antar posisi bot dan diamond
    base_dist = abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)
    min_dist = base_dist
    best_teleporter = None

    teleporters = [o for o in board.game_objects if o.type == "TeleportGameObject"]
    for tp_a in teleporters:
        for tp_b in teleporters:
            if tp_a is not tp_b:
                dist = (
                    abs(pos1.x - tp_a.position.x) + abs(pos1.y - tp_a.position.y) + 1 +
                    abs(tp_b.position.x - pos2.x) + abs(tp_b.position.y - pos2.y)
                )
                if dist < min_dist:
                    min_dist = dist
                    best_teleporter = tp_a

    return min_dist, best_teleporter

def heuristic(self, pos1: Position, pos2: Position, board: Board, ignore_red=False,
time_left=None) -> float:
    # jarak terbaik antara posisi bot dan diamond
    min_dist, _ = self.get_best_path(pos1, pos2, board)

    # Hitung weight objek pada posisi tujuan
    weight = 0
    for obj in board.game_objects:
        if obj.position.x == pos2.x and obj.position.y == pos2.y:
            weight = self.get_weight(obj, board, ignore_red)

    # total weight cluster diamond
    cluster_value = self.get_cluster_value(pos2, board, ignore_red)
    weight += cluster_value * 0.5

    # semakin sedikit waktu , maka bot akan mencari diamond yang lebih dekat dengan base
    time_penalty = 0
    if time_left:
        time_penalty = max(0, min_dist * self.base_time_penalty * (1000 / time_left))

    return min_dist - weight + time_penalty

#logika bot untuk kembali ke base
def gobaselogic(self, bot: GameObject, board: Board, nearest_diamond_dist: int) -> bool:
    base = bot.properties.base
    steps_to_base = abs(bot.position.x - base.x) + abs(bot.position.y - base.y)
    time_left = int(bot.properties.milliseconds_left / 1000)
    return (
        bot.properties.diamonds >= 3 and nearest_diamond_dist > steps_to_base
        or steps_to_base >= time_left
        or bot.properties.diamonds >= self.inv_full
    )

```

```

def next_move(self, board_bot: GameObject, board: Board) -> Tuple[int, int]:
    self.position = board_bot.position
    base = board_bot.properties.base
    inventory = board_bot.properties.diamonds
    time_left = board_bot.properties.milliseconds_left

    # Filter diamond
    diamonds = [o for o in board.game_objects if o.type == "DiamondGameObject"]
    ignore_red = inventory >= 4
    if ignore_red:
        diamonds = [d for d in diamonds if getattr(d.properties, "points", 1) == 1]

    if not diamonds:
        return self.move_towards(base)

    # Pilih target dengan heuristic yang sudah memperhitungkan waktu
    nearest = min(diamonds,
                  key=lambda d: self.heuristic(self.position, d.position,
                                                board, ignore_red, time_left))
    nearest_dist, best_teleporter = self.get_best_path(self.position, nearest.position, board)

    # Cek apakah perlu kembali ke base
    if self.gobaselogic(board_bot, board, nearest_dist):
        return self.move_towards(base)

    # Gunakan jika dengan menggunakan teleporter lebih dekat ke diamond
    if best_teleporter:
        return self.move_towards(best_teleporter.position)

    # Bergerak ke diamond terdekat
    return self.move_towards(nearest.position)

def move_towards(self, target: Position) -> Tuple[int, int]:
    #bergerak ke arah target
    dx = target.x - self.position.x
    dy = target.y - self.position.y
    if abs(dx) > abs(dy):
        return (1 if dx > 0 else -1, 0)
    elif dy != 0:
        return (0, 1 if dy > 0 else -1)
    return (1, 0)

```


4.2 Implementasi Algoritma Greedy dalam Bot Diamonds Menggunakan Pseudocode

Dalam implementasi program bot permainan diamonds menggunakan bahasa pemrograman python kami membuat 7 method yang dibagi menjadi 2 kategori fungsi yaitu method untuk menghitung heuristik dan fungsi untuk memilih arah untuk bergerak berdasarkan heuristik terendah. Dalam setiap method, implementasi strategi algoritma greedy kami buat dengan pendekatan fungsional dan kami akan membahas implementasi algoritma program dalam pseudocode dimulai dari fungsi perhitungan heuristik dan fungsi logika pemilihan arah gerak.

- Kategori Fungsi Perhitungan Heuristik

Method kategori ini bertujuan untuk melakukan perhitungan heuristik yang memberikan opsi kepada fungsi logika pemilihan arah gerak untuk memilih arah gerak bot yang paling efisien berdasarkan algoritma greedy

- Method get_weight

Fungsi get_weight(objek, papan, abaikan_merah=False):

- Jika objek adalah "DiamondGameObject":
 - Ambil nilai "points" dari properti objek, jika tidak ada set ke 1
 - Jika abaikan_merah bernilai True dan poin objek adalah 2:
 - Return 0 (jika diamond merah harus diabaikan)
 - Jika poin objek adalah 2:
 - Return 2 (nilai diamond merah)
 - Jika tidak:
 - Return 1 (nilai diamond biru)
- Jika objek adalah tipe "DiamondButtonGameObject":
 - Cari base dari bot
 - Jika base ditemukan:
 - Tentukan objek-objek yang berada dalam jarak 4 unit dari base
 - Hitung total weight kumulatif dari semua diamond yang berada di sekitar base
 - Jika total weight diamond di sekitar base kurang dari atau sama dengan 2:
 - Return 3 (nilai DiamondButtonGameObject karena ada diamond dekat base)
 - Jika tidak:
 - Return 0 (tidak perlu diambil)
- Jika objek bukan tipe "DiamondGameObject" atau "DiamondButtonGameObject":
 - Return 0 (objek tidak bernilai)

- Method get_cluster_value

Fungsi get_cluster_value(posisi, papan, abaikan_merah):

- cluster = daftar kosong
- // inisialisasi list/array untuk menghitung weight kumulatif sekitar diamond
- Untuk setiap objek o dalam papan.game_objects:
 - Jika objek o adalah "DiamondGameObject":
 - Jika selisih posisi x dan posisi y dari objek dengan posisi bot \leq radius cluster:

```

    ■ Tambahkan objek o ke dalam cluster
// Memeriksa apakah posisi diamond dalam radius cluster

    • total_weight = 0
    • Untuk setiap objek dalam cluster:
        ○ total_weight = total_weight + hasil dari get_weight(o, papan,
          abaikan_merah)

// Menghitung dan mengembalikan total weight cluster dengan return
    • Return total_weight

```

○ Method get_best_path

```

Fungsi get_best_path(pos1, pos2, papan):
    • base_dist = selisih posisi x1 dan x2 + selisih posisi y1 dan y2
      //(x1,y1) (x2,y2) posisi base dan posisi bot
    • min_dist = base_dist
    • best_teleporter = None
// Menghitung jarak antara posisi bot dan diamond

    • teleporters = daftar teleporters dari papan.game_objects yang bertipe
      "TeleportGameObject"
// Mencari semua teleporters yang ada di papan

    • Untuk setiap teleporter tp_a dalam teleporters:
    • Untuk setiap teleporter tp_b dalam teleporters:
    • Jika tp_a bukan tp_b:
        ○ dist = selisih posisi x1 dan posisi x tp_a + selisih posisi y1 dan
          posisi y tp_a + 1 + selisih absolut posisi x tp_b dan posisi x2 +
          selisih absolut posisi y tp_b dan posisi y2
          //mencari jarak teleportasi dengan diamonds berdasarkan selisih
          jarak bot + teleport + jarak ke diamond

    • Jika dist < min_dist:
        ○ min_dist = dist
        ○ best_teleporter = tp_a
//Jika jarak teleportasi lebih kecil dari jarak yang dihitung sebelumnya
mengembalikan jarak minimum dan teleporter terbaik yang ditemukan

    • return min_dist, best_teleporter

```

○ Method heuristic

```

Fungsi heuristic(posisi1, posisi2, papan, abaikan_merah=False,
waktu_tersisa=None):
    • min_dist, _ ← get_best_path(posisi1, posisi2, papan)
// Menghitung jarak terbaik antara posisi bot dan diamond

    • weight = 0
    • Untuk setiap objek o dalam papan.game_objects:

```

- Jika posisi objek o sama dengan posisi2:
 - `weight = get_weight(o, papan, abaikan_merah)`
- // Menghitung weight objek di posisi tujuan
- `cluster_value = get_cluster_value(posisi2, papan, abaikan_merah)`
- `weight = weight + (cluster_value * 0.5)`
- // Menghitung total weight cluster diamond di sekitar posisi tujuan
- `time_penalty = 0`
- Jika waktu_tersisa ada:
 - `time_penalty = maksimum(0, min_dist * base_time_penalty * (1000 / waktu_tersisa))`
- // Menghitung penalti waktu
- `Return min_dist - weight + time_penalty`
- // Mengembalikan nilai heuristik

◦ Method gobaseologic

Fungsi gobaseologic(bot, papan, nearest_diamond_dist):

- `base = bot.properties.base`
- // Mendapatkan posisi base dari bot
- `steps_to_base = selisih posisi x bot dan posisi x base + selisih posisi y bot dan posisi y base`
- // Menghitung langkah yang diperlukan untuk kembali ke base
- `time_left = konversi waktu bot dari milidetik ke detik (bot.properties.milliseconds_left / 1000)`
- // Mendapatkan waktu yang tersisa dan konversi waktu dari milidetik ke detik
- Jika `bot.properties.diamonds >= 3` dan `nearest_diamond_dist > steps_to_base` atau `steps_to_base >= time_left` atau `bot.properties.diamonds >= inv_full`:
 - Return True
 - Jika tidak, Return False
- // Mengembalikan True jika salah satu kondisi terpenuhi

• Kategori Fungsi Logika Pemilihan Arah Gerak

◦ Method next_move

Fungsi next_move(board_bot, papan):

- `posisi_bot = board_bot.position`
- `base = board_bot.properties.base`
- `inventory = board_bot.properties.diamonds`
- `waktu_tersisa = board_bot.properties.milliseconds_left`
- // Mengambil posisi bot dan properti lainnya
- `diamonds = daftar objek diamond di papan.game_objects`
- `abaikan_merah = inventory >= 4`
- Jika `abaikan_merah`:
 - `diamonds = filter hanya diamond yang bernilai 1 poin (diamond biru)`

```
// Filter diamond yang ada di papan

    • Jika tidak ada diamonds:
      ◦ Return hasil move_towards(base)
// Jika tidak ada diamond yang bisa diambil, bergerak ke base

    • nearest = diamond dengan nilai heuristik terkecil, dihitung dengan fungsi
      heuristic
// Pilih diamond terdekat menggunakan heuristic yang mempertimbangkan waktu

    • nearest_dist, best_teleporter ← get_best_path(posisi_bot, nearest.position,
      papan)
// Menghitung jarak terbaik dan teleporter terbaik untuk menuju diamond terdekat

    • Jika gobaseLogic(board_bot, papan, nearest_dist):\
      ◦ Return hasil move_towards(base)
// Cek apakah perlu kembali ke base

    • Jika best_teleporter:
      ◦ Return hasil move_towards(best_teleporter.position)
// Jika menggunakan teleporter lebih cepat ke diamond, bergerak ke teleporter

    • Return hasil move_towards(nearest.position)
//bergerak ke diamond terdekat
```

○ Method move_towards

```
Fungsi move_towards(target):
    • dx ← target.x - posisi.x
    • dy ← target.y - posisi.y
// Menghitung perbedaan posisi antara target dan posisi bot

    • Jika selisih dx lebih besar dari selisih absolut dy:
      ◦ Jika dx lebih besar dari 0:
        ■ Return (1, 0) // Bergerak ke kanan (positif sumbu x)
      ◦ Jika tidak:
        ■ Return (-1, 0) // Bergerak ke kiri (negatif sumbu x)
// Jika perbedaan posisi x lebih besar dari y, bergerak di sumbu x

    • Jika dy tidak sama dengan 0:
      ◦ Jika dy lebih besar dari 0:
        ■ Return (0, 1) // Bergerak ke atas (positif sumbu y)
      ◦ Jika tidak:
        ■ Return (0, -1) // Bergerak ke bawah (negatif sumbu y)
// Jika perbedaan posisi y tidak nol, bergerak di sumbu y

    • Return (1, 0)
// Jika tidak ada perbedaan pada sumbu x atau y, bergerak ke kanan secara default
```

4.3 Penjelasan Struktur Data dalam Program Bot Diamonds

Untuk mengimplementasikan logika bot yang telah ditentukan, beberapa struktur data dasar diperlukan untuk merepresentasikan keadaan permainan dan entitas di dalamnya. Struktur data ini memungkinkan bot untuk menyimpan, mengakses, dan memanipulasi informasi yang dibutuhkan untuk pengambilan keputusan:

1. Representasi Peta/Board:

- Board adalah struktur data yang mewakili papan permainan tempat objek-objek (seperti diamond dan bot) berinteraksi. Board memiliki berbagai atribut yang menyimpan informasi tentang objek-objek yang ada di dalam permainan serta status terkini permainan.
- Direpresentasikan sebagai sebuah grid 2D di mana setiap sel dapat berisi informasi tentang objek yang ada di koordinat tersebut. Ukuran grid yaitu 15x15.

2. Representasi Objek Game:

Diperlukan struktur atau kelas untuk setiap jenis objek penting dalam permainan:

- Diamond:
 - Atribut: posisi (koordinat x,y), tipe (red diamond, blue diamond), base_weight (nilai poin: 2 untuk merah, 1 untuk biru).
 - final_weight: Didapat dengan memperhitungkan base_weight ditambah bonus dari diamond berdekatan.
- Base:
 - Atribut: posisi (koordinat x,y), owner_id (untuk mengidentifikasi kepemilikan base).
- Teleporter:
 - Atribut: posisi_masuk (koordinat x,y), posisi_keluar (koordinat x,y).
- Red Button:
 - Atribut: posisi (koordinat x,y).

3. Status Bot:

Bot adalah entitas dalam permainan yang berusaha mengumpulkan diamond sebanyak-banyaknya. Setiap bot memiliki atribut seperti inventory, base, posisi, dan strategi gerakan.

Atribut:

- posisi_sekarang (koordinat x,y).
- isi_inventory (jumlah diamond yang dibawa).
- sisa_waktu (Sisa waktu permainan dalam satuan detik).rr

Bot akan menggunakan informasi yang ada pada Board dan GameObject untuk menentukan langkah yang akan diambil, berdasarkan algoritma Greedy yang telah diterapkan.

4. Inventory:

Inventory adalah struktur data yang digunakan untuk menyimpan diamond yang telah dikumpulkan oleh bot. Inventory memiliki kapasitas tertentu, dan jika kapasitas penuh, bot harus kembali ke base untuk menyimpan diamond tersebut.

Atribut:

- max_capacity: Kapasitas maksimum inventory bot.
- current_capacity: Jumlah diamond yang sedang dibawa oleh bot.

5. Daftar Teleporter & Red Button: Mirip dengan daftar diamond, diperlukan daftar untuk menyimpan informasi teleporter dan red button yang ada di peta.

6. Efisiensi dan Pengelolaan Waktu:

Untuk mengoptimalkan kinerja bot, terutama dalam hal efisiensi waktu, struktur data akan mempertimbangkan waktu yang tersisa serta posisi diamond dan base bot. Jika waktu hampir

habis, bot akan memilih untuk kembali ke base atau menghindari perjalanan yang memakan waktu terlalu lama.

7. Fungsi Seleksi dan Heuristik:

Algoritma Greedy mengandalkan fungsi seleksi untuk menentukan langkah terbaik berdasarkan kriteria yang sudah ditentukan, seperti jarak terdekat atau poin tertinggi. Fungsi seleksi bertujuan untuk memaksimalkan jumlah diamond yang dikumpulkan oleh bot dalam waktu yang tersisa.

- Fungsi Seleksi: Memilih langkah terbaik berdasarkan kriteria jarak(distance), poin(weight), atau density(cluster).
- Heuristik: Digunakan untuk mengevaluasi keputusan yang diambil oleh bot dan menentukan langkah yang akan dilakukan selanjutnya.

4.4 Analisis Desain Solusi Algoritma

Pada permainan Diamonds, bot yang dikembangkan bertujuan untuk mengumpulkan diamond sebanyak-banyaknya dengan menggunakan Algoritma Greedy. Algoritma ini bertujuan untuk membuat keputusan pada setiap langkah berdasarkan pilihan yang terbaik secara lokal, tanpa mempertimbangkan konsekuensi jangka panjang. Dalam hal ini, bot memilih diamond untuk dikumpulkan dengan menggunakan berbagai kriteria, seperti kedekatannya dengan bot dan nilai poin yang diberikan oleh diamond tersebut.

1. Desain Solusi Algoritma Greedy

Desain solusi algoritma Greedy pada permainan Diamonds didasarkan pada beberapa langkah dan prinsip yang memanfaatkan elemen-elemen yang ada dalam permainan. Berikut adalah analisis lebih mendalam mengenai solusi algoritma yang diterapkan:

1.1. Himpunan Kandidat

Himpunan kandidat pada permainan ini mencakup berbagai objek yang ada di dalam permainan yang bisa menjadi target langkah bot. Objek-objek ini antara lain:

- Diamond: Merupakan objek utama yang harus dikumpulkan oleh bot. Terdapat dua jenis diamond, yaitu diamond merah yang bernilai 2 poin dan diamond biru yang bernilai 1 poin.
- Teleporters: Objek yang memungkinkan bot berpindah dari satu lokasi ke lokasi lain dengan cepat, mempengaruhi pilihan langkah bot.
- Red Button: Tombol yang digunakan untuk mereset jumlah dan mengatur ulang posisi diamond pada board.
- Base: Lokasi tempat bot menyimpan diamond yang telah dikumpulkan. Setelah mencapai base, bot akan mendapatkan skor berdasarkan diamond yang disimpan.

1.2. Himpunan Solusi

Himpunan solusi pada algoritma Greedy ini terdiri dari langkah-langkah yang diambil bot dalam permainan untuk mengumpulkan diamond sebanyak mungkin. Solusi terdiri dari:

- Langkah menuju diamond yang terdekat: Bot memilih diamond berdasarkan kriteria jaraknya, berusaha mengumpulkan sebanyak mungkin diamond dalam waktu terbatas.
- Penggunaan Teleporters: Bot akan memilih untuk menggunakan teleporter apabila memberikan keuntungan dalam hal efisiensi jumlah langkah yang perlu diambil(jarak) dibandingkan dengan bergerak biasa.

- Kembali ke Base: Setelah bot mencapai kapasitas maksimum dalam inventory, bot harus kembali ke base untuk menyimpan diamond yang telah dikumpulkan.
- 1.3. Fungsi Solusi

Fungsi solusi mengevaluasi apakah langkah yang diambil sudah memenuhi tujuan permainan, yaitu mengumpulkan diamond sebanyak-banyaknya. Beberapa kriteria yang digunakan dalam fungsi solusi adalah:

 - Total diamond yang terkumpul: Apakah jumlah diamond yang berhasil dikumpulkan telah memenuhi target atau kapasitas inventory bot.
 - Kembali ke base tepat waktu: Jika langkah yang diambil bot untuk mengumpulkan diamond menyebabkan inventory bot penuh, bot harus segera kembali ke base untuk menyimpan diamond sebelum waktu permainan habis.
- 1.4. Fungsi Seleksi (Selection Function)

Fungsi seleksi bertugas untuk memilih langkah terbaik bagi bot pada setiap langkahnya. Dalam hal ini, bot menggabungkan konsep Greedy by Distance, Greedy by Points dan Greedy by Density untuk mengukur heuristik yang kemudian digunakan untuk menjadi opsi gerakan yang akan diseleksi.

Penjelasan ketiga konsep greedy yaitu:

 - Greedy by Distance: Memilih diamond berdasarkan jarak terdekat dari posisi bot. Tujuan dari pendekatan ini adalah untuk meminimalkan jumlah langkah yang diambil.
 - Greedy by Points: Memilih diamond dengan poin tertinggi terlebih dahulu.
 - Greedy by Density: Memilih diamond yang memiliki rasio poin terbesar (density) berdasarkan diamond yang saling berdekatan(cluster), untuk mengoptimalkan pengumpulan diamond dengan waktu yang terbatas.
- 1.5. Fungsi Kelayakan (Feasibility Function)

Fungsi kelayakan memeriksa apakah langkah yang dipilih oleh bot dapat dilakukan dengan mempertimbangkan beberapa faktor, seperti:

 - Ketersediaan waktu: Jika bot telah mengumpulkan diamond dan kapasitas inventory penuh, bot harus memutuskan apakah cukup waktu untuk kembali ke base.
- 1.6. Fungsi Objektif (Objective Function)

Fungsi objektif bertujuan untuk memaksimalkan jumlah diamond yang dikumpulkan oleh bot selama permainan. Fungsi ini memprioritaskan pilihan diamond berdasarkan density terbesar (poin per jarak), serta pengambilan diamond merah dengan nilai lebih tinggi jika kondisi memungkinkan.
- 2. Evaluasi Desain Solusi
 - 2.1. Kelebihan Algoritma Greedy
 - Cepat dalam pengambilan keputusan: Bot selalu memilih langkah yang terbaik pada saat itu tanpa perlu mencari solusi optimal secara keseluruhan, sehingga bot bisa bergerak cepat.
 - Fleksibilitas: Algoritma ini dapat dengan mudah disesuaikan untuk memilih antara berbagai pendekatan, seperti Greedy by Distance, Greedy by Points, dan Greedy by Density.
 - 2.2. Kekurangan Algoritma Greedy
 - Potensi tidak optimal: Karena bot memilih solusi terbaik secara lokal tanpa mempertimbangkan konsekuensi jangka panjang, algoritma ini tidak selalu

menghasilkan solusi yang optimal.

- Ketergantungan pada pengaturan awal: Beberapa kondisi, seperti teleporters yang tidak dipertimbangkan dengan baik, dapat menyebabkan bot memilih langkah yang kurang efisien.

4.5 Kasus Pengujian

Untuk memastikan bahwa algoritma Greedy yang diterapkan pada bot permainan Diamonds dapat bekerja secara optimal dalam berbagai situasi, berikut adalah beberapa kasus pengujian yang mencakup kondisi-kondisi unik yang mungkin terjadi selama permainan. Pengujian ini bertujuan untuk mengevaluasi apakah strategi yang digunakan berhasil mencapai hasil yang diinginkan, yaitu mengumpulkan diamond sebanyak-banyaknya dengan mempertimbangkan waktu yang terbatas, posisi diamond, dan potensi risiko seperti pertemuan dengan bot musuh.

- Kasus Pengujian 1: Bot Mengambil Diamond Terdekat
Bot berada di papan permainan dengan beberapa diamond yang tersebar. Bot akan mengutamakan untuk mengumpulkan diamond dengan jarak terdekat tanpa memperhatikan nilai poinnya. Pengujian dilakukan untuk melihat apakah bot dapat memaksimalkan jumlah diamond yang dikumpulkan dalam waktu yang terbatas dengan memilih diamond yang terdekat.
 - Hasil yang Diharapkan: Bot akan mengumpulkan diamond dengan lebih cepat karena memilih diamond yang terdekat. Namun, bot mungkin akan mengabaikan diamond merah yang lebih menguntungkan jika terlalu jauh dari posisinya.
- Kasus Pengujian 2: Bot Mengambil Diamond dengan Nilai Poin Tertinggi
Bot akan memprioritaskan untuk mengumpulkan diamond dengan nilai poin tertinggi terlebih dahulu, meskipun diamond tersebut lebih jauh dari posisi bot. Pengujian dilakukan untuk melihat apakah bot dapat memaksimalkan poin yang diperoleh dengan memilih diamond dengan nilai tertinggi, meskipun mengharuskan bot untuk menempuh jarak yang lebih jauh.
 - Hasil yang Diharapkan: Bot mengumpulkan diamond merah terlebih dahulu meskipun jaraknya lebih jauh, sehingga berpotensi memperoleh lebih banyak poin. Namun, bisa saja tidak efisien dalam segi jarak dan waktu.
- Kasus Pengujian 3: Bot Menggunakan Red Button
Bot berada di dekat Red Button, dan menggunakan tombol tersebut untuk me-reset posisi diamond di papan permainan. Pengujian dilakukan untuk melihat apakah penggunaan Red Button dapat meningkatkan peluang bot dalam mengumpulkan lebih banyak diamond setelah regenerasi posisi diamond.
 - Hasil yang Diharapkan: Bot menggunakan Red Button dengan cerdas untuk mengatur ulang posisi diamond, memungkinkan pengumpulan diamond lebih banyak setelah regenerasi.
- Kasus Pengujian 4: Bot Menghadapi Waktu yang Tinggal Sedikit.
Dalam kondisi ini, bot harus membuat keputusan yang efisien dan cepat untuk mengumpulkan diamond sebanyak mungkin dalam waktu yang tersisa sebelum permainan berakhir. Bot harus mengevaluasi pilihan berdasarkan kecepatan pengumpulan diamond dan mempertimbangkan apakah kembali ke base dapat dilakukan.
 - Bot akan mengutamakan pengumpulan diamond yang dapat dijangkau dalam waktu terbatas. Jika inventory bot sudah penuh atau waktu untuk kembali ke base lebih sedikit daripada waktu untuk mengumpulkan diamond, bot akan memilih untuk kembali ke base dan menyimpan diamond yang sudah terkumpul.

- Kasus Pengujian 5: Bot Menghadapi Kapasitas Inventory Penuh
Bot mengumpulkan diamond hingga inventory penuh, dan harus segera kembali ke base untuk menyimpan diamond tersebut. Pengujian dilakukan untuk melihat apakah bot dapat kembali ke base tepat waktu dan menyimpan diamond yang telah dikumpulkan sebelum inventory penuh.
 - Hasil yang Diharapkan: Bot berhasil kembali ke base dengan diamond yang telah dikumpulkan dan mendapatkan poin dari diamond yang disimpan.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan laporan tugas besar mengenai implementasi Algoritma Greedy pada bot permainan Diamonds, dapat disimpulkan bahwa algoritma ini merupakan pendekatan yang efisien untuk memecahkan masalah optimasi dalam pengumpulan diamond. Dengan menerapkan berbagai solusi greedy seperti Greedy by Distance, Greedy by Weight, dan Greedy by Density, bot dapat membuat keputusan yang bertujuan untuk memaksimalkan jumlah diamond yang dikumpulkan.

Namun, perlu dicatat bahwa algoritma Greedy tidak selalu memberikan solusi optimal dari semua pilihan solusi yang memungkinkan. Hal ini mengingat bahwa algoritma ini hanya memilih langkah terbaik berdasarkan kondisi saat itu tanpa mempertimbangkan dampak keputusan jangka panjang. Meskipun demikian, algoritma ini sangat efektif dalam konteks waktu terbatas dan pengambilan keputusan cepat, yang merupakan sifat dari permainan Diamonds.

5.2 Saran

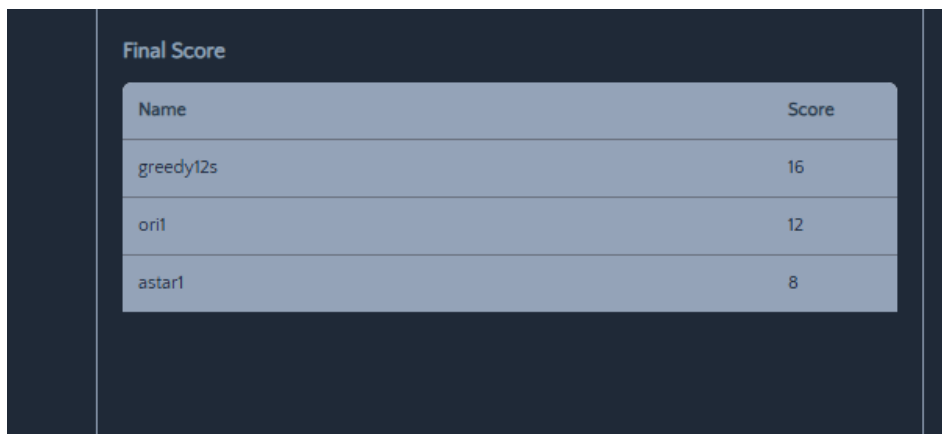
- **Optimasi Algoritma Greedy:**
Untuk meningkatkan performa bot, sebaiknya dilakukan optimasi lebih lanjut pada algoritma Greedy by Density dengan mempertimbangkan faktor tambahan seperti pergerakan musuh. Hal ini bisa membantu bot menghadapi situasi yang lebih dinamis. Penulis tidak memasukkan algoritma pergerakan musuh dikarenakan hasilnya yang sangat tidak konsisten. Jadi, penulis mencari jalan aman dan tidak mengimplementasikannya ke dalam algoritma bot.
- **Peningkatan Pengujian:**
Sebaiknya dilakukan pengujian lebih lanjut pada berbagai kondisi ekstrem, seperti bot yang menghadapi waktu terbatas atau bot yang bertemu musuh. Dengan pengujian lebih mendalam, strategi bot dapat lebih disempurnakan untuk menghindari kesalahan keputusan yang mungkin mengarah pada kerugian poin.
- **Fitur Adaptif:**
Menambahkan fitur adaptif pada bot yang memungkinkan penyesuaian strategi selama permainan berdasarkan berbagai kondisi yang didapat dari testing yang lebih lanjut.

Lampiran

Repository Github: <https://github.com/Protoflicker/greedyalgorithmbot-diamonds>

Link Youtube:

Screenshot Testing Bot (bot semuanya menggunakan logika yang sama ([greedy12.py](#)))



Name	Score
greedy12s	16
ori1	12
astar1	8



Name	Score
ori1	16
astar1	13
greedy12s	10
greedyred1	10

Daftar Pustaka

Munir, R. (2021). *Algoritma Greedy Bagian 1*. Bahan Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB. Retrieved from [M 04 - Algoritma Greedy Bag1.pdf](#)

Munir, R. (2021). *Algoritma Greedy Bagian 2*. Bahan Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB. Retrieved from [M 05 - Algoritma Greedy Bag2.pdf](#)

Munir, R. (2022). *Algoritma Greedy Bagian 3*. Bahan Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB. Retrieved from [M 06 - Algoritma Greedy Bag3.pdf](#)

Maulidevi, N. U., & Munir, R. (2021). *Penentuan Rute (Route/Path Planning)*. Bahan Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB. Retrieved from [M 14 - Route Planning Bag1.pdf](#)

Maulidevi, N. U., & Munir, R. (2021). *Penentuan Rute (Route/Path Planning)*. Bahan Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika ITB. Retrieved from [M 15 - Route Planning Bag2.pdf](#)

Tubes1_JadiMesin. (2025). *Tugas Besar: Algoritma Greedy pada Bot Permainan Diamonds*. GitHub Repository. Retrieved from https://github.com/PanjiSri/Tubes1_JadiMesin.git.

Tubes1_susugratis. (2025). *Tugas Besar: Implementasi Algoritma Greedy pada Bot Permainan Diamonds*. GitHub Repository. Retrieved from https://github.com/akmalrmn/Tubes1_susugratis.