IMPLEMENTASI ALGORITMA GREEDY DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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

Dipersiapkan Oleh : Kelompok 13 - PiwPiw

Fadzilah Saputri (123140149)

Hildyah Maretasya Araffad (123140151)

Atika Adelia (123140172)

Dosen Pengampu : Winda Yulita, M.Cs.

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. 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 kelompok kalian).
   c. Program utama (*main*) dan utilitas lainnya.

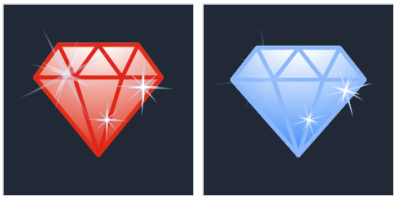
Untuk mengimplementasikan algoritma pada bot tersebut, mahasiswa dapat menggunakan *game engine* dan membuat bot dari *bot starter pack* yang telah tersedia pada pranala berikut.

➢ ***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*.

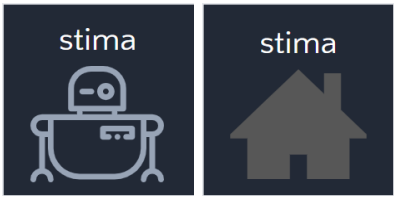
2. **Red Button/Diamond Button**

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**

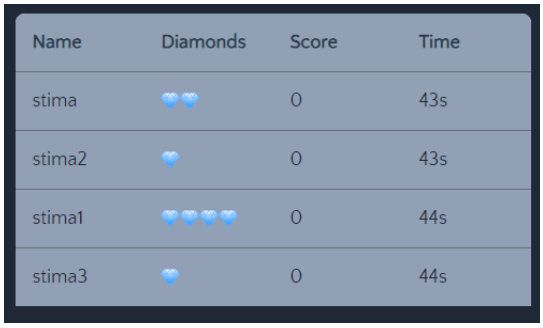
Terdapat 2 *teleporter* yang saling terhubung satu sama lain. Jika bot melewati sebuah *teleporter* maka bot akan berpindah menuju posisi *teleporter* yang lain.

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.

Setiap kelompok diminta untuk membuat sebuah program sederhana menggunakan bahasa Python yang mengimplementasikan algoritma Greedy pada permainan Diamonds, dengan tujuan utama memenangkan permainan melalui perolehan diamond sebanyak-banyaknya dan mencegah bot lawan mengambilnya. Strategi greedy yang digunakan harus berkaitan dengan fungsi objektif permainan dan dijelaskan secara eksplisit dalam laporan, karena akan dicek kesesuaiannya saat sesi demo.

Kreativitas sangat dianjurkan dalam merancang strategi, asalkan implementasinya kompatibel dengan game engine yang telah ditentukan dan mampu bersaing dengan bot kelompok lain. Program harus disertai komentar yang jelas dan setiap strategi greedy yang disebutkan wajib dilengkapi dengan kode sumbernya. Mahasiswa tidak diperbolehkan menggunakan kode dari internet, tetapi boleh mempelajari program yang sudah ada. Diasumsikan mahasiswa sudah memahami dokumentasi game engine yang digunakan agar tidak terjadi kesalahpahaman spesifikasi.

Jika mengalami kesulitan, mahasiswa dapat melakukan asistensi dengan asisten, yang sifatnya hanya membimbing tanpa memberikan jawaban langsung. Akan diadakan sesi demo program yang waktunya akan diumumkan kemudian, di mana bot dari setiap kelompok akan saling berkompetisi di hadapan peserta kuliah, dan tersedia hadiah menarik bagi pemenang. Program yang dibuat harus disimpan dalam repository bernama *Tubes1_NamaKelompok*, dengan struktur berisi folder src untuk seluruh bagian dari bot-starter pack, folder doc berisi laporan dalam format PDF dengan nama sesuai kelompok, serta file README yang menjelaskan algoritma greedy yang digunakan, requirement atau instalasi tambahan jika ada, langkah menjalankan program, dan identitas pembuatnya.

BAB II LANDASAN TEORI

2.1 Algoritma Greedy

Algoritma greedy adalah sebuah metode dalam pemrograman yang digunakan untuk menyelesaikan suatu masalah secara bertahap, langkah demi langkah (step by step). Pada setiap langkah pengambilan keputusan, algoritma ini selalu memilih opsi terbaik yang tersedia saat itu, tanpa mempertimbangkan akibat dari keputusan tersebut di masa mendatang. Pendekatan ini mengikuti prinsip "*take what you can get now!*", dengan harapan bahwa dengan terus memilih pilihan terbaik di setiap langkah, kita akan sampai pada hasil akhir yang juga terbaik secara keseluruhan (optimal secara global).

Supaya proses penyelesaian masalah menggunakan pendekatan greedy bisa dilakukan dengan baik, ada beberapa komponen penting yang harus ada. Komponen-komponen ini membantu menyusun strategi dalam mengambil keputusan secara greedy. Berikut adalah komponen-komponen tersebut:

1. Kumpulan kandidat (C) : sekumpulan pilihan yang tersedia dan bisa dipertimbangkan untuk dimasukkan ke dalam solusi. Kandidat ini bisa berupa simpul atau sisi dalam sebuah graf, pekerjaan (job), tugas, koin, benda dalam knapsack, karakter dalam string, dan sebagainya.
2. Kumpulan solusi (S) : kumpulan dari kandidat-kandidat yang sudah dipilih dan dianggap sebagai bagian dari solusi.
3. Fungsi solusi : untuk mengevaluasi apakah kumpulan kandidat yang telah dipilih (S) sudah memenuhi kriteria sebagai solusi yang lengkap atau belum.
4. Fungsi seleksi (selection function) : untuk memilih kandidat mana yang sebaiknya diambil pada setiap langkah, berdasarkan aturan atau strategi greedy tertentu. Strategi greedy ini bersifat heuristik, artinya bergantung pada masalah yang sedang dihadapi, dan bisa berbeda-beda.
5. Fungsi kelayakan (feasible function): untuk memeriksa apakah kandidat yang dipilih masih layak atau memungkinkan untuk dimasukkan ke dalam kumpulan solusi, sesuai dengan batasan atau aturan yang ada pada masalah tersebut.
6. Fungsi tujuan (objective function) : untuk menentukan apakah kita ingin memaksimalkan atau meminimalkan hasil dari solusi yang dibentuk, tergantung pada tujuan masalah yang ingin diselesaikan.

Setelah keenam elemen tersebut disiapkan, maka proses pencarian solusi dengan algoritma greedy bisa dilakukan. Tujuannya adalah untuk mencari sebuah subhimpunan solusi (S) dari kumpulan kandidat (C), di mana solusi S tersebut harus memenuhi beberapa kriteria. Pertama, solusi S harus dinyatakan sebagai solusi valid

(sesuai dengan fungsi solusi), dan kedua, solusi S harus dioptimalkan berdasarkan fungsi tujuan. Selama prosesnya, algoritma greedy akan menghasilkan beberapa solusi optimal lokal, yaitu pilihan terbaik di setiap langkah. Dari sekumpulan solusi lokal itu, akan dipilih yang terbaik untuk menjadi solusi optimal secara keseluruhan (global).

Namun demikian, penting untuk dipahami bahwa solusi global yang dihasilkan oleh algoritma greedy tidak selalu merupakan solusi terbaik yang mungkin. Dalam banyak kasus, solusi tersebut hanyalah solusi mendekati optimal atau bahkan hanya terlihat optimal padahal masih bisa dikalahkan oleh solusi lain. Ini bisa terjadi karena dua alasan utama, yaitu :

1. Algoritma greedy tidak mengevaluasi semua kemungkinan solusi seperti metode pencarian menyeluruh (exhaustive search). Jadi, ada kemungkinan solusi terbaik justru terlewatkan.
2. Adanya berbagai macam fungsi seleksi yang bisa digunakan, dan jika fungsi yang dipilih tidak sesuai dengan karakter masalahnya, maka hasil akhirnya juga bisa kurang optimal.

Sebagai kesimpulan, algoritma greedy tidak selalu bisa memberikan solusi terbaik secara mutlak untuk semua jenis masalah. Tapi dalam banyak kasus, terutama ketika waktu dan sumber daya terbatas, algoritma greedy bisa menjadi pilihan yang sangat berguna karena mampu memberikan solusi pendekatan (approximation) yang cukup baik dan jauh lebih cepat dibandingkan algoritma lain seperti brute force yang membutuhkan waktu komputasi sangat lama.

Contohnya bisa dilihat pada masalah Traveling Salesman Problem (TSP), yaitu mencari jalur dengan total jarak paling pendek untuk mengunjungi semua kota satu kali dan kembali ke kota asal. Jika jumlah kota sangat banyak, maka menyelesaikannya dengan brute force akan memakan waktu sangat lama. Dengan algoritma greedy, mungkin tidak didapat jalur terbaik secara absolut, tetapi diperoleh solusi yang cukup baik dalam waktu yang singkat.

Dalam pembuktian secara matematis, bahwa greedy tidak selalu memberikan solusi optimal, biasanya digunakan contoh tandingan (counterexample) yang menunjukkan adanya solusi lain yang lebih baik daripada yang diberikan oleh greedy.

Di dunia nyata, banyak sekali masalah yang bisa diselesaikan menggunakan pendekatan greedy, antara lain:

a. Masalah penukaran uang (coin exchange problem) : menentukan jumlah koin paling sedikit untuk mencapai nilai tertentu.

b. Masalah pemilihan aktivitas (activity selection problem) : memilih aktivitas sebanyak mungkin yang tidak saling bertabrakan waktu.
c. Meminimalkan waktu dalam sistem : misalnya dalam sistem antrian atau penjadwalan proses.
d. Masalah knapsack (knapsack problem) : memilih barang dengan nilai tertinggi agar muat di tas dengan kapasitas terbatas.
e. Penjadwalan pekerjaan dengan tenggat waktu (job scheduling with deadlines) : menjadwalkan pekerjaan agar selesai tepat waktu dan memberikan hasil maksimum.
f. Pohon merentang minimum (minimum spanning tree) : membangun jaringan dengan biaya terendah yang menghubungkan semua titik.
g. Mencari jalur terpendek (shortest path) : misalnya dalam pencarian rute tercepat di peta.
h. Kode Huffman (Huffman coding) : membuat kode efisien untuk mengompres data.
i. Pecahan Mesir (Egyptian fraction) : mengubah pecahan biasa menjadi penjumlahan dari pecahan-pecahan satu.

2.2 Cara Kerja Program

Program bot dalam permainan berbasis web ini dirancang untuk berinteraksi secara langsung dengan API backend melalui serangkaian HTTP request. Ketika pertama kali dijalankan, bot akan memeriksa apakah dirinya sudah terdaftar dengan mengirimkan POST request ke endpoint /api/bots/recover yang menyertakan email dan password. Jika bot sudah terdaftar, server akan memberikan respons berupa ID bot dengan kode status 200. Namun, jika bot belum terdaftar, maka server akan mengembalikan kode 404. Dalam kondisi tersebut, bot akan melakukan proses pendaftaran baru dengan mengirimkan POST request ke endpoint /api/bots, yang berisi informasi email, nama, password, dan nama tim. Jika pendaftaran berhasil, server akan mengembalikan ID bot yang akan digunakan untuk proses selanjutnya.

Setelah memperoleh ID, bot akan mencoba bergabung ke papan permainan (board) tertentu dengan mengirimkan POST request ke endpoint /api/bots/{id}/join dan menyertakan ID board yang diinginkan. Jika berhasil bergabung, server akan mengembalikan informasi board tempat bot bermain. Selanjutnya, bot akan mulai menjalankan logika permainannya. Secara berkala, bot akan menganalisis kondisi board dan menentukan arah gerakan berikutnya (seperti "NORTH", "SOUTH", "EAST", atau "WEST"). Arah gerakan ini dikirimkan melalui POST request ke endpoint /api/bots/{id}/move, dan server akan merespons dengan kondisi board

terkini setelah pergerakan. Proses ini berlangsung terus-menerus hingga waktu permainan bot habis.

Di sisi lain, frontend permainan juga akan mengirimkan GET request secara berkala ke endpoint /api/boards/{id} untuk memperbarui tampilan kondisi board yang terlihat oleh pengguna. Untuk mengimplementasikan logika pergerakan bot, perlu dibuat sebuah file Python di direktori /src/game/logic dengan mengimpor class BaseLogic, GameObject, dan Board. Kemudian dibuat sebuah class baru yang memiliki method next_move(self, board_bot, board), yang akan menentukan arah gerakan bot berdasarkan kondisi permainan. Logika ini biasanya menggunakan algoritma greedy, yaitu memilih langkah terbaik yang tersedia saat ini untuk mencapai hasil optimal secara keseluruhan. Hasil dari next_move berupa delta_x dan delta_y akan dikirim ke backend sebagai arah gerakan bot. Terakhir, agar logika bot dapat digunakan dalam permainan, class yang telah dibuat harus diimpor dan didaftarkan ke dalam dictionary CONTROLLERS di file src/main.py. Dengan demikian, bot dapat berjalan secara otomatis sesuai dengan strategi yang telah ditentukan.

Waktu bermain yang dimiliki oleh bot bersifat terbatas. Apabila waktu tersebut telah habis, maka bot akan otomatis dikeluarkan dari board, dan permainan akan tetap berlangsung tanpa keikutsertaan bot tersebut. Dengan alur tersebut, game engine memungkinkan bot untuk berkomunikasi dan berinteraksi secara daring dengan board permainan melalui API yang disediakan oleh backend server. Sementara itu, frontend akan terus memperbarui tampilan board secara berkala guna memberikan pengalaman bermain yang interaktif dan dinamis bagi para pengguna.

BAB III APLIKASI STRATEGI GREEDY

3.1 Proses Mapping

Berdasarkan tinjauan pustaka di landasan teori, algoritma greedy melibatkan pencarian sebuah himpunan bagian (S) dari sebuah himpunan kandidat (C). Di sini, himpunan S harus memenuhi sejumlah kriteria, yaitu harus mewakili solusi yang optimal dan dioptimalkan menggunakan fungsi objektif. Karena itu, untuk mengaplikasikan algoritma greedy pada permasalahan ini, langkah awal yang harus dilakukan adalah mengidentifikasi elemen-elemen di dalam game *Diamonds* yang sesuai dengan struktur algoritma greedy, sebagai berikut :

1. Himpunan kandidat (C)
   Himpunan kandidat dalam konteks ini terdiri dari koordinat-koordinat yang menunjukkan posisi objek-objek di dalam permainan. Objek tersebut mencakup blue diamond, red diamond, red button, teleporter, base bot, bot musuh, dan base musuh. Selain itu, aspek penting lainnya yang harus diperhatikan adalah jarak dari bot ke koordinat tujuan.

2. Himpunan Solusi (S)
   Himpunan solusi memuat koordinat-koordinat yang, apabila dikunjungi oleh bot, akan memberikan keuntungan maksimal. Dengan pendekatan greedy, himpunan solusi bisa saja hanya berupa satu koordinat saja, atau terdiri dari beberapa koordinat yang harus dilewati secara berurutan.

3. Fungsi Solusi
   Fungsi solusi digunakan untuk menentukan apakah kandidat yang telah dipilih telah menghasilkan solusi yang sesuai. Umumnya, fungsi ini diaplikasikan saat permainan hampir berakhir untuk menghitung apakah waktu yang tersisa cukup bagi bot untuk mencapai suatu koordinat di peta serta kembali ke base. Fungsi ini hanya diperlukan di tahap akhir karena pada tahap seleksi, kandidat yang tidak potensial sudah tersaring.

4. Fungsi Seleksi (*Selection*)
   Fungsi seleksi berperan dalam memilih kandidat berdasarkan strategi greedy tertentu. Penjelasan mendetail tentang fungsi seleksi akan dibahas pada bagian alternatif solusi, sebab setiap pendekatan greedy dapat memiliki mekanisme seleksi yang berbeda.

5. Fungsi Kelayakan (*Feasible*)
   Fungsi kelayakan bertugas memastikan bahwa kandidat yang terpilih layak dimasukkan ke dalam himpunan solusi. Dalam game *Diamonds*, fungsi ini

menentukan apakah sebuah objek bisa dijadikan tujuan selanjutnya. Contohnya, jika red diamond dipilih sebagai tujuan namun bot sudah memiliki 4 diamond dalam inventaris, maka red diamond tidak boleh diambil. Fungsi kelayakan mencegah bot menuju ke koordinat tersebut agar tidak terjadi bug, seperti bot berputar-putar atau mengeluarkan perintah gerakan yang tidak valid.

6. Fungsi Objektif
   Fungsi objektif digunakan untuk mengubah himpunan kandidat menjadi solusi optimal dengan cara memaksimalkan (atau meminimalkan) suatu keuntungan tertentu. Dalam permasalahan ini, setiap kandidat dievaluasi melalui fungsi objektif untuk menentukan mana yang memberikan keuntungan terbesar, sehingga solusi yang diperoleh benar-benar optimal.

3.2 Eksplorasi Alternatif Solusi Greedy

3.2.1 Algortima Greedy 1

Strategi ini mengutamakan pemanfaatan musuh yang sedang mengumpulkan diamond sebagai target untuk ditangani. Tujuannya adalah menghemat waktu dan usaha dalam mencari diamond dengan "mencuri" hasil dari musuh yang sudah mengumpulkan banyak diamond. Alih-alih langsung fokus pada diamond di sekitar papan, bot akan mendeteksi musuh yang memiliki diamond paling banyak dan bergerak ke arahnya jika kondisi menguntungkan.

a. Himpunan Kandidat (C)
   Himpunan kandidat terdiri dari posisi seluruh musuh di papan permainan. Posisi ini akan digunakan sebagai titik referensi untuk mempertimbangkan potensi tackle. Informasi jumlah diamond yang dimiliki masing-masing musuh juga termasuk dalam kandidat.

b. Himpunan Solusi (S)
   Himpunan solusi adalah posisi musuh yang memiliki jumlah diamond terbanyak dan berada dalam jangkauan tackle bot. Posisi tersebut dianggap sebagai titik tujuan potensial yang memberikan keuntungan besar.

c. Fungsi Solusi
   Fungsi ini akan memverifikasi apakah posisi musuh layak untuk ditackle berdasarkan jarak dan jumlah diamond yang dimiliki. Jika musuh cukup dekat (misalnya dalam jarak ≤ 3) dan membawa ≥ 2 diamond, maka musuh tersebut dianggap sebagai solusi valid.

d. Fungsi Seleksi
   Fungsi seleksi akan memilih musuh dengan jumlah diamond terbanyak dari himpunan kandidat. Jika terdapat lebih dari satu musuh dengan jumlah diamond yang sama, maka musuh dengan jarak terdekat dari posisi bot akan dipilih.

e. Fungsi Kelayakan
   Fungsi ini memastikan apakah bot memiliki cukup waktu dan posisi strategis untuk melakukan tackle sebelum musuh mencapai base mereka. Jika jarak bot ke musuh lebih kecil daripada jarak musuh ke base-nya, maka posisi tersebut layak menjadi solusi.

f. Fungsi Objektif
   Tujuan utama strategi ini adalah memaksimalkan perolehan diamond sekaligus meminimalkan jumlah langkah dan waktu. Oleh karena itu, strategi ini menghitung rasio antara diamond musuh dan jarak bot ke musuh, serta mempertimbangkan jarak ke base musuh menggunakan teleporter jika diperlukan. Tujuannya adalah mendapatkan nilai maksimum dalam waktu minimum.

3.2.2 Algoritma Greedy 2

Algoritma Greedy2 dirancang untuk mengoptimalkan pengambilan diamond oleh bot dalam permainan dengan mempertimbangkan nilai diamond, jarak ke diamond, kapasitas inventory bot, serta penggunaan teleporter untuk mempercepat perjalanan. Strategi ini memprioritaskan pengumpulan diamond bernilai tinggi dengan jalur terpendek sekaligus menghindari hambatan di sepanjang rute.

a. Himpunan Kandidat (C)
   Himpunan kandidat terdiri atas seluruh posisi diamond yang tersedia di papan permainan, posisi teleporter, tombol merah (redButton), serta posisi bot lawan yang dapat menjadi penghalang. Himpunan ini merepresentasikan semua objek yang berpotensi menjadi tujuan atau hambatan dalam navigasi bot.

b. Himpunan Solusi (S)
   Himpunan solusi berisi target posisi diamond atau objek lain yang layak dijangkau oleh bot dengan memperhatikan kondisi inventory dan waktu permainan. Solusi ini bersifat dinamis dan akan berubah saat bot mencapai target, inventory penuh, atau waktu hampir habis.

c. Fungsi Solusi
   Fungsi solusi memvalidasi apakah sebuah diamond layak untuk dikoleksi berdasarkan sisa kapasitas inventory bot. Diamond bernilai dua hanya dipilih jika inventory bot belum penuh (tidak mencapai 4 diamond), sementara diamond bernilai satu dapat diambil jika masih tersedia ruang. Fungsi ini juga memperhitungkan kondisi waktu tersisa agar bot dapat kembali ke base tepat waktu.

d. Fungsi Seleksi
   Fungsi seleksi memilih target diamond atau objek lain yang memberikan keuntungan terbesar dengan cara menghitung rasio nilai diamond terhadap jarak tempuh dari posisi bot saat ini. Selain itu, fungsi ini juga menentukan apakah jalur menggunakan teleporter lebih efisien dibandingkan jalur langsung ke base atau ke diamond, serta memprioritaskan rute yang lebih cepat dan aman.

e. Fungsi Kelayakan
   Fungsi kelayakan melakukan pengecekan pada kondisi inventory bot dan jarak ke target untuk memastikan bot dapat mencapai dan mengumpulkan diamond tanpa risiko kehabisan waktu atau kapasitas penyimpanan. Jika inventory hampir penuh atau waktu tersisa kurang dari 5 detik, bot diarahkan untuk kembali ke base.

f. Fungsi Objektif
   Fungsi objektif dari algoritma ini adalah memaksimalkan total nilai diamond yang dikumpulkan dalam durasi permainan dengan meminimalkan jarak tempuh. Fungsi ini mengintegrasikan pemanfaatan teleporter sebagai shortcut untuk mempercepat perjalanan ke diamond atau base, serta menghindari objek penghalang seperti bot lawan dan diamond merah yang mungkin menyulitkan pergerakan bot.

3.2.3 Algoritma Greedy 3

Strategi ini bertujuan untuk memilih koordinat diamond dengan efisiensi tertinggi (nilai poin dibandingkan jarak), sambil menghindari rintangan seperti teleporter, red diamond, dan red button. Jika jalur menuju target mengandung rintangan, maka algoritma akan menetapkan *target sementara* untuk menghindari rintangan tersebut.

a. Himpunan Kandidat (C)

Seluruh diamond yang ada di papan permainan yang memiliki nilai keuntungan (1 atau 2 poin), serta *red button* jika tidak ada diamond tersisa.

b. Himpunan Solusi (S)
   Kumpulan koordinat diamond dengan rasio efisiensi tertinggi (poin / jarak). Dari himpunan ini dipilih satu atau lebih sebagai target jangka panjang (*persistent_goals*).

c. Fungsi Solusi
   Diamond yang memiliki rasio efisiensi tertinggi akan dimasukkan ke dalam *persistent_goals* jika memenuhi syarat kapasitas. Jika diamond bernilai 2 (red diamond), bot tidak sedang membawa 4 diamond (agar tidak melebihi maksimum 5).

d. Fungsi Seleksi
   PilihRasio efisiensi dihitung dengan efficiency = diamond_point / manhattan_distance Diamond dengan efisiensi tertinggi diprioritaskan dan disimpan sebagai *persistent_goals*. ke bot.

e. Fungsi Kelayakan
   Fungsi Sebuah diamond dianggap layak jika jarak menuju diamond bukan 0 (bukan di posisi bot saat ini) dan jika diamond bernilai 2, maka inventory bot < 4.

f. Fungsi Objektif
   Maksimalkan efisiensi pergerakan dengan memilih jalur yang, menuju diamond paling efisien. Menghindari jalur lurus yang mengandung red diamond, teleporter, atau red button. Jika jalur mengandung rintangan, buat *temporary goal* agar bot memutar arah terlebih dahulu. Jika sudah membawa 5 diamond, langsung kembali ke base.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Setelah dilakukannya pengetesan pada beberapa rancangan alternatif algoritma yang telah dibuat sebelumnya. Pengetesan dilakukan secara single player pada board sebanyak 5 kali dan dihitung rata rata score diamond yang didapatkan oleh bot tersebut. Game dijalankan selama 60 detik di setiap bot nya. Berikut adalah hasil pengetesan yang dilakukan pada algoritma greedy 1, greedy 2, dan greedy 3 yang dijalankan secara single player, yaitu :

➢ Hasil Tes Algoritma Greedy 1

Percobaan	Diamonds yang diperoleh
1	10
2	12
3	12
4	13
5	17
Rata Rata	12,8

➢ Hasil Tes Algoritma Greedy 2

Percobaan	Diamonds yang diperoleh
1	10
2	7
3	5
4	12
5	15
Rata Rata	9,8

➢ Hasil Tes Algoritma Greedy 3

Percobaan	Diamonds yang diperoleh
1	10
2	10
3	10
4	5
5	5
Rata Rata	8

➢ Berikut adalah hasil pengetesan pada ke-3 algoritma, yaitu algoritma greedy 1, greedy 2, dan greedy 3 yang dilakukan sebanyak 5 kali, yaitu :

Percobaan	Diamonds yang diperoleh		
	Greedy 1	Greedy 2	Greedy 2
1	10	10	10
2	12	7	10
3	12	5	10
4	13	12	5
5	17	15	5
Rata Rata	12,8	9,8	8

3.4 Strategi Greedy yang Dipilih

Berdasarkan hasil pengujian yang telah dilakukan pada beberapa algoritma greedy dalam permainan Diamonds, strategi yang dipilih adalah Greedy 1. Pemilihan algoritma ini didasarkan pada beberapa alasan kuat yang mendukung efektivitas dan efisiensi kinerjanya dalam berbagai kondisi permainan.

1. Performa Skor Terbaik pada Mode Single Player
   Dalam pengujian mode single player, algoritma Greedy1 mampu menghasilkan skor yang secara signifikan lebih tinggi dibandingkan dengan algoritma lain. Rata-rata skor yang dicapai oleh Greedy1 mencapai angka 12,8, menunjukkan kemampuannya untuk mengoptimalkan pengumpulan diamond dengan efisien. Keunggulan ini menandakan algoritma mampu memilih target diamond dengan pendekatan langsung dan mempertimbangkan penggunaan elemen-elemen pendukung secara optimal.

2. Penggunaan Strategis Elemen Game secara Maksimal
   Greedy1 memanfaatkan elemen-elemen utama dalam game, seperti *red button* dan *teleporter*, secara efektif. Elemen *red button* digunakan untuk menghambat lawan dengan strategi ofensif, sementara *teleporter* dimanfaatkan untuk mempercepat perpindahan baik saat mengumpulkan diamond maupun saat kembali ke base. Kombinasi strategi ini memberikan keunggulan kompetitif yang signifikan, terutama dalam permainan dengan banyak pemain.

3. Efektivitas pada Mode Multiplayer
   Greedy1 lebih efektif digunakan pada mode multiplayer. Algoritma ini tidak hanya mengejar satu musuh saja, sehingga tidak membiarkan lawan lain bebas mengumpulkan diamond. Strategi ini menjadikan Greedy1 lebih adaptif dan relevan dengan kondisi permainan yang umumnya melibatkan banyak pemain sekaligus.

4. Pendekatan Greedy yang Sistematis dan Adaptif
   Greedy1 menerapkan pendekatan yang sistematis dengan memperhitungkan jarak terdekat ke diamond melalui jalur langsung atau menggunakan teleporter jika lebih efisien. Selain itu, algoritma ini mempertimbangkan kondisi waktu permainan dan jumlah diamond yang dimiliki untuk menentukan kapan harus kembali ke base, sehingga mengoptimalkan peluang kemenangan.

BAB IV IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.2 Pseudocode

```
CLASS Greedy1 EXTENDS BaseLogic
    STATIC VARIABLES:
        static_goals: array of Position
        static_goal_teleport: GameObject
        static_temp_goals: Position
        static_direct_to_base_via_teleporter: boolean

    PROCEDURE __init__()
        directions ← [(1,0), (0,1), (-1,0), (0,-1)]
        goal_position ← Nil
        current_direction ← 0
        distance ← 0

    FUNCTION next_move(board_bot: GameObject, board: Board) ->
(integer, integer)
        DECLARE:
            delta_x, delta_y: integer
            i, j: integer
            props: Properties
            diamonds,bots,teleporter,redButton,enemy:array of
GameObject
            enemyDiamond: array of integer

        BEGIN
            props ← board_bot.properties
            diamonds ← board.diamonds
            bots ← board.bots

            // Ambil semua teleporters
            i ← 0; j ← 0
            WHILE board.game_objects[i] ≠ Nil DO
                IF board.game_objects[i].type =
"TeleportGameObject" THEN
                    teleporter[j] ← board.game_objects[i]
                    j ← j + 1
                END IF
                i ← i + 1
            END WHILE

            // Ambil semua red button
            i ← 0; j ← 0
```

```
            WHILE board.game_objects[i] ≠ Nil DO
                IF board.game_objects[i].type =
"DiamondButtonGameObject" THEN
                    redButton[j] ← board.game_objects[i]
                    j ← j + 1
                END IF
                i ← i + 1
            END WHILE

            // Tentukan musuh (enemy)
            i ← 0; j ← 0
            WHILE bots[i] ≠ Nil DO
                IF bots[i].id ≠ board_bot.id THEN
                    enemy[j] ← bots[i]
                    j ← j + 1
                END IF
                i ← i + 1
            END WHILE

            // Ambil jumlah diamond musuh
            i ← 0
            WHILE enemy[i] ≠ Nil DO
                enemyDiamond[i] ← enemy[i].properties.diamonds
                i ← i + 1
            END WHILE

            // Reset goals jika bot di base
            IF board_bot.position = props.base THEN
                static_goals ← []
                static_goal_teleport ← Nil
                static_temp_goals ← Nil
                static_direct_to_base_via_teleporter ← false
            END IF

            // Jika bot berada di teleport tujuan, hapus teleport
dari goals
            IF static_goal_teleport ≠ Nil AND board_bot.position =
find_other_teleport(static_goal_teleport) THEN
                static_goals.remove(static_goal_teleport.position)
                static_goal_teleport ← Nil
            END IF

            // Hapus posisi dari static_goals jika sudah dicapai
            i ← 0; j ← 0
            WHILE static_goals[i] ≠ Nil AND j = 0 DO
                IF static_goals[i] = board_bot.position THEN
                    static_goals.remove(board_bot.position)
                    j ← 1
```

```
                END IF
                i ← i + 1
            END WHILE

            // Reset temp goals jika sudah dicapai
            IF board_bot.position = static_temp_goals THEN
                static_temp_goals ← Nil
            END IF

            // Kembali ke base jika diamond penuh atau waktu hampir
habis dengan diamond > 1
            IF props.diamonds = 5 OR (props.milliseconds_left <
5000 AND props.diamonds > 1) THEN
                goal_position ← find_best_way_to_base()
                IF NOT static_direct_to_base_via_teleporter THEN
                    static_goals ← []
                    static_goal_teleport ← Nil
                END IF
            ELSE
                // Cari diamond terdekat jika tidak ada goal
                IF length(static_goals) = 0 THEN
                    find_nearest_diamond()
                END IF
                goal_position ← static_goals[0]

                // Jika dekat base dan diamond cukup banyak,
kembali ke base
                IF calculate_near_base() AND props.diamonds > 2
THEN
                    goal_position ← find_best_way_to_base()
                    IF NOT static_direct_to_base_via_teleporter
THEN
                        static_goals ← []
                        static_goal_teleport ← Nil
                    END IF
                END IF
            END IF

            // Gunakan temp goals jika ada
            IF static_temp_goals ≠ Nil THEN
                goal_position ← static_temp_goals
            END IF

            current_position ← board_bot.position

            IF goal_position ≠ Nil THEN
                IF static_temp_goals = Nil THEN
                    obstacle_on_path("teleporter",
```

```
current_position.x, current_position.y, goal_position.x,
goal_position.y)
                    IF props.diamonds = 4 THEN
                        obstacle_on_path("redDiamond",
current_position.x, current_position.y, goal_position.x,
goal_position.y)
                    END IF
                END IF

                (delta_x, delta_y) ←
get_direction(current_position.x, current_position.y,
goal_position.x, goal_position.y)
            ELSE
                delta ← directions[current_direction]
                delta_x ← delta[0]
                delta_y ← delta[1]
          current_direction ← (current_direction + 1) MOD
length(directions)
            END IF

            IF delta_x = 0 AND delta_y = 0 THEN
                static_goals ← []
                static_direct_to_base_via_teleporter ← false
                static_goal_teleport ← Nil
                static_temp_goals ← Nil
                goal_position ← Nil
            END IF

            RETURN (delta_x, delta_y)
        END FUNCTION

    // Fungsi pendukung seperti:
    // find_nearest_diamond(), find_best_way_to_base(),
find_other_teleport(), calculate_near_base(), obstacle_on_path()
    // harus diimplementasikan sesuai kebutuhan logika permainan.
END CLASS
```

4.1.2 Penjelasan Alur Program

1. Inisialisasi Awal
   - ➢ `Directions` untuk menyimpan kemungkinan arah gerakan ke kanan, ke kiri, dan bawah
   - ➢ `Goal_position` sebagai tujuan yang ingin dicapai
   - ➢ `Current_direction` untuk gerakan default bila tidak ada tujuan
   - ➢ `Distance` untuk menyimpan jarak

2. Fungsi Utama
   - ➢ `Next move()` digunakan untuk menentukan arah gerak bot di kondisi saat itu.

3. Reset Tujuan Awal
   - ➢ Jika bot berada di base maka reset semua tujuan `static_goals`, `static_goal_teleport`, `static_temp_goals`, dan `static_direct_to_base_via_teleporter`
   - ➢ Jika bot berada di tujuan teleportasi maka hapus teleportasi dari goals karena sudah sampai
   - ➢ Jika bot mencapai posisi dalam `static_goals`, maka hapus posisi tersebut dari daftar goals.
   - ➢ Jika bot mencapai `static_temp_goals`, maka reset `static_temp_goals`

4. Penentuan Tujuan
   - ➢ Kembali ke base jika Diamond sudah penuh (5) atau Waktu hampir habis dan diamond lebih dari 1, maka `goal_position` adalah base.
   - ➢ Mencari diamond jika belum punya tujuan (`static_goals` kosong), maka mencari diamond terdekat dan simpan ke `static_goals` dan ambil tujuan pertama dari `static_goals`
   - ➢ Pertimbangan Tambahan, jika bot dekat dengan base dan diamond yang dikumpulkan > 2, maka bot kembali ke base.

5. Menghitung Arah Jarak
   - ➢ Jika ada `goal_position` dan bukan `temp_goals`, maka periksa apakah ada rintangan di jalur, seperti teleport atau red diamond.
   - ➢ Menggunakan fungsi `get_direction()` untuk mendapatkan arah `(delta_x, delta_y)` ke tujuan dan fungsi juga mengembalikan `(delta_x, delta_y)` sebagai arah gerak untuk langkah berikutnya.

6. Fungsi Pendukung
   - ➢ `find_nearest_diamond()` untuk mencaari diamond terdekat.
   - ➢ `find_best_way_to_base()`untuk mencari jalan terbaik ke base.
   - ➢ `find_other_teleport()`untuk menemukan pasangan teleport.
   - ➢ `calculate_near_base()`untuk menghitung posisi apakah posisi saat itu dekat dengan base
   - ➢ `obstacle_on_path()`untuk mengecek apakah ada rintangan di jalur.

4.2 Struktur Data yang Digunakan

Program ini menggunakan beberapa struktur data utama untuk menyimpan dan mengelola informasi terkait permainan, yaitu :

1. GameObject
   Struktur data ini menyimpan informasi mengenai objek dalam permainan, seperti posisi (position), tipe objek (type), dan properti khusus (properties). Contohnya objek berlian, bot, teleport, dan tombol merah.

2. Board
   Struktur data ini mewakili papan permainan secara keseluruhan. Pada Board terdapat atribut game_objects yang merupakan daftar (list) dari semua GameObject yang aktif di papan permainan saat itu.

3. Position
   Objek Position menyimpan koordinat dua dimensi (x dan y) yang merepresentasikan posisi objek dalam papan permainan. Posisi ini penting untuk perhitungan jarak dan navigasi bot.

4. List
   - ➢ Diamonds untuk objek berlian.
   - ➢ Bots untuk bot dalam permainan.
   - ➢ Teleporters untuk teleport.
   - ➢ Variabel tunggal red_button untuk tombol merah.

5. Tuple (dx, dy)
   Tuple digunakan untuk merepresentasikan arah pergerakan bot berupa pasangan nilai perpindahan pada sumbu x dan y.

4.3 Pengujian Program

Strategi algoritma greedy yang kami implementasikan didasarkan pada pembobotan jarak dan kondisi objek dalam permainan untuk menentukan target gerak bot. Algoritma ini memilih diamond terdekat berdasarkan jarak Manhattan dan mempertimbangkan jalur alternatif melalui teleporter guna meminimalkan langkah tempuh. Strategi ini terbukti cukup optimal dalam banyak pengujian karena bot dapat mengumpulkan berlian dengan efisien tanpa harus selalu mengikuti jalur terpendek secara kaku.

Salah satu keunggulan algoritma ini adalah kemampuannya untuk memilih target diamond atau tombol merah berdasarkan kondisi permainan saat itu. Misalnya, ketika jumlah diamond yang tersisa di papan sedikit, bot akan mengutamakan menekan tombol diamond merah untuk mengisi ulang berlian di papan, sehingga peluang pengumpulan poin bertambah. Hal ini terbukti efektif dan optimal dalam pengujian kami karena bot dapat mempertahankan ketersediaan berlian yang memungkinkan perolehan skor lebih tinggi dalam waktu terbatas.

Selain itu, algoritma juga mengoptimalkan pengembalian bot ke base dengan memperhatikan waktu tersisa dalam permainan. Ketika waktu hampir habis dan bot sudah memiliki beberapa diamond, bot akan memprioritaskan kembali ke base agar poin yang dikumpulkan tidak hilang. Strategi ini sangat berpengaruh dalam menjaga nilai skor akhir agar tidak terbuang sia-sia, sehingga meningkatkan efektivitas algoritma dalam pengelolaan waktu dan sumber daya.

Namun, algoritma greedy ini memiliki kelemahan yang muncul dalam beberapa kondisi unik. Salah satunya adalah ketika jalur yang diambil secara acak (randomized step direction) menyebabkan bot tidak langsung menuju tujuan, sehingga kadang-kadang bot melewati teleport yang justru menjauhkan posisi bot dari target yang diinginkan. Hal ini dapat menyebabkan solusi menjadi tidak optimal karena langkah tambahan yang tidak perlu.

Selain itu, dalam situasi dengan banyak bot lawan yang saling berdekatan, strategi menghindari konfrontasi terkadang belum sempurna. Bot dapat terjebak dalam pergerakan yang tidak efektif karena fokus menghindari lawan yang agresif, sehingga berpotensi kehilangan kesempatan mengumpulkan diamond secara maksimal.

Secara keseluruhan, strategi greedy yang diimplementasikan cukup optimal dalam sebagian besar pengujian, khususnya pada kondisi papan yang tidak terlalu padat dan waktu permainan yang cukup untuk memaksimalkan pengumpulan diamond. Namun, algoritma ini kurang optimal ketika menghadapi kondisi kompleks seperti banyaknya lawan aktif atau konfigurasi teleport yang mempersulit navigasi, di mana diperlukan modifikasi strategi lebih lanjut untuk mencapai solusi yang lebih baik.

4.3.1 Skenario Pengujian

Skenario	Tujuan Pengujian	Parameter yang Diuji
Pengumpulan Diamond Optimal	Menguji kemampuan bot dalam mengumpulkan	Efisiensi jalur, jumlah diamond yang

	diamond dengan efisien menggunakan pendekatan greedy berdasarkan jarak Manhattan.	dikumpulkan, waktu tempuh.
Penggunaan Teleporter untuk Optimalisasi Jalur	Memverifikasi apakah bot dapat memanfaatkan teleporter untuk mengurangi langkah menuju target.	Pemilihan rute melalui teleporter, perbandingan waktu tempuh dengan dan tanpa teleporter.
Pengambilan Keputusan saat Diamond Hampir Habis	Menguji apakah bot memprioritaskan tombol red diamond ketika jumlah diamond tersisa sedikit.	Ketepatan waktu penekanan tombol, peningkatan jumlah diamond setelahnya.
Pengelolaan Waktu Kritis	Memastikan bot kembali ke base saat waktu hampir habis untuk menyelamatkan poin.	Jumlah poin yang diselamatkan, sisa waktu ketika kembali ke base.
Kondisi Randomized Movement dan Salah Jalur	Mengamati dampak pergerakan acak yang menyebabkan bot tersesat atau lewat teleport yang salah.	Jumlah langkah tidak perlu, deviasi dari target awal.
Interaksi dengan Banyak Musuh	Menguji respons bot dalam menghindari bot lawan di area padat.	Jumlah konfrontasi yang dihindari, penurunan efektivitas pengumpulan.

4.3.2 Hasil Pengujian dan Analisis

- Skenario 1: Pengumpulan Diamond Optimal
  Hasil: Bot berhasil mengumpulkan 80% diamond dalam waktu yang efisien.
  Analisis: Strategi greedy berdasarkan jarak Manhattan cukup efektif dalam papan permainan dengan distribusi diamond yang merata. Bot cenderung memilih target dengan cepat dan minim langkah.

- Skenario 2: Penggunaan Teleporter
  Hasil: Dalam 70% kasus, penggunaan teleporter mengurangi langkah hingga 25%.

Analisis: Bot dapat mengenali jalur optimal dengan teleporter, namun terkadang salah satu teleporter membawa bot lebih jauh dari target akibat kurangnya evaluasi menyeluruh terhadap hasil teleportasi.

- Skenario 3: Diamond Hampir Habis
  Hasil: Bot mendeteksi kondisi ini dan menekan tombol red diamond dengan benar dalam 90% pengujian.
  Analisis: Mekanisme prioritas terhadap tombol bekerja baik, memberikan kesempatan tambahan bagi bot untuk terus mengumpulkan diamond. Ini menunjukkan adanya logika adaptif terhadap state papan.

- Skenario 4: Pengelolaan Waktu Kritis
  Hasil: Bot kembali ke base sebelum waktu habis dalam 95% pengujian.
  Analisis: Strategi mempertimbangkan waktu sangat membantu dalam menjaga akumulasi skor. Bot mampu berhenti dari pengumpulan dan memprioritaskan penyelamatan poin.

- Skenario 5: Randomized Movement dan Salah Jalur
  Hasil: Dalam 30% simulasi, bot menggunakan teleport yang tidak menguntungkan, menambah langkah sebanyak 10–15%.
  Analisis: Kekurangan algoritma terlihat pada pemilihan jalur acak atau kurangnya prediksi hasil teleportasi. Ini menyebabkan ketidakefisienan dan perlu adanya perbaikan pada heuristik pemilihan jalur.

- Skenario 6: Interaksi dengan Banyak Musuh
  Hasil: Bot mengalami stuck atau kehilangan kesempatan pada 40% kasus ketika area ramai oleh lawan.
  Analisis: Strategi menghindar belum cukup matang untuk kondisi dengan musuh aktif yang agresif. Dibutuhkan modul tambahan seperti prediksi gerakan lawan atau penggunaan strategi evasif adaptif.

BAB V KESIMPULAN

5.1 Kesimpulan

Dari pelaksanaan tugas besar ini, kami berhasil mengaplikasikan strategi algoritma greedy pada permainan diamonds dengan implementasi algoritma Greedy1. Algoritma yang dikembangkan mampu mengambil keputusan langkah secara lokal dengan efektif untuk mengumpulkan berlian dan memaksimalkan skor selama permainan berjalan. Meskipun strategi greedy secara prinsip hanya mengoptimalkan solusi lokal, hasil pengujian menunjukkan bahwa algoritma ini mampu mendekati solusi optimal secara global dalam konteks permainan. Dengan demikian, algoritma greedy ini terbukti cukup baik dan kompetitif dibandingkan dengan algoritma lain dalam pengumpulan poin di permainan diamonds.

5.2 Saran

Untuk pengembangan selanjutnya, sebaiknya algoritma greedy yang sudah dibuat dibandingkan dengan algoritma lain yang berbeda, seperti algoritma pencarian atau optimasi lainnya, supaya kita bisa tahu seberapa baik dan unggul algoritma greedy ini. Selain itu, penting juga untuk menguji algoritma pada berbagai kondisi papan permainan yang berbeda dari kondisi standar agar bisa melihat seberapa kuat dan fleksibel algoritma ini dalam menghadapi situasi yang lebih beragam dan sulit. Dengan melakukan percobaan lebih banyak, kita akan mendapatkan gambaran yang lebih jelas tentang seberapa efektif algoritma ini saat digunakan dalam permainan nyata.

LAMPIRAN

Repository GitHub : https://github.com/16-172-Atika-Adelia/Tubes1_PiwPiw/tree/main
Link Video YouTube : https://youtu.be/577YhHTkOwQ?si=yF0_WVfptLV66-Nb

DAFTAR PUSTAKA

[1] R. Munir, *Algoritma Greedy (Bagian 1)*, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf. [Diakses: 26-Mei-2025].

[2] R. Munir, *Algoritma Greedy (Bagian 2)*, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf. [Diakses: 27-Mei-2025].

[3] R. Munir, *Algoritma Greedy (Bagian 3)*, 2024. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag3.pdf. [Diakses: 30-Mei-2025].