

IMPLEMENTASI ALGORITMA THE MASTER SIFU DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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



Oleh: Kelompok 5 (Skibidi)

M.Daffa Hakim Matondang 123140002

Febrian Valentino 123140034

Ola Anggela Rosita 123140042

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

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INDUSTRI
INSTITUT TEKNOLOGI SUMATERA
2025**

DAFTAR ISI

BAB I DESKRIPSI TUGAS.....	1
BAB II LANDASAN TEORI.....	3
2.1 Dasar Teori.....	3
2.2 Cara Kerja Program.....	6
BAB III APLIKASI STRATEGI GREEDY.....	8
3.1 Proses Mapping.....	8
3.2 Eksplorasi Alternatif Solusi Greedy.....	9
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy.....	11
3.4 Strategi Greedy yang Dipilih.....	12
BAB IV IMPLEMENTASI DAN PENGUJIAN.....	14
4.1 Implementasi Algoritma Greedy.....	14
4.2 Struktur Data yang Digunakan.....	20
4.3 Pengujian Program.....	27
BAB V KESIMPULAN DAN SARAN.....	37
5.1 Kesimpulan.....	37
5.2 Saran.....	37
LAMPIRAN.....	39
DAFTAR PUSTAKA.....	40

BAB I

DESKRIPSI TUGAS

Penerapan Algoritma Greedy pada Pembuatan Bot Permainan Diamonds

Besarnya tugas ini adalah membuat sebuah bot yang dapat bermain secara otomatis dalam permainan Diamonds. Diamonds adalah sebuah challenge pemrograman yang mempertandingkan kecerdasan bot-bot dari berbagai programmer dalam mengumpulkan berlian (diamond) sebanyak-banyaknya di dalam sebuah arena virtual.

Pada kompetisi ini setiap bot akan diberikan kesempatan untuk berkompetisi di sebuah arena yang sudah disediakan, dan tujuan utamanya adalah mengumpulkan diamond yang tersebar di seluruh arena. Tetapi saja Diamonds bukan merupakan permainan yang mudah diatasi, karena terdapat banyak sekali obstacle atau rintangan yang harus dilewati serta kompetisi secara real time dengan bot-bot berlawanan.

Salah satu faktor yang membuat bot ini menarik adalah adanya sistem collide, dimana bot dapat saling “menabrak” satu sama lain. Ketika hal ini terjadi, bot yang tertabrak akan menderita kehilangan semua diamond dan item yang ada di inventory dan akan dikirim pulang ke base, sedangkan bot yang menabrak akan mendapatkan gold tersebut. Ini semua menciptakan dinamika permainan yang membuat bot tidak hanya berfokus pada diamond, mereka harus bersikap waspada terhadap bot lawan.

Berdasarkan apa yang telah dibahas sebelumnya, dalam permainan ini ada beberapa added value yang bisa dikelola dengan cukup eksploratif. Permainan diamond memiliki dua jenis, yaitu diamond biru dan diamond merah. Terdapat pula red button yang akan mengacak posisi diamond di papan, dan teleporter yang dapat memindahkan bot ke lokasi lain.

Hal mendasar yang bisa dibilang paling esensial dalam penyelesaian tugas ini adalah yang berkaitan dengan algoritma Greedy. Algoritma ini seharusnya diimplementasikan di dalam solusi yang diajukan. Dalam konteks ini, mereka berhadapan dengan masalah yang bernama

Diamonds dan mendirikan sebuah strategi yang tertera dalam algoritma greedy. Setiap keputusan perlu menjadi keputusan terbaik yang diambil sebelumnya, contohnya adalah kapan bot harus memulai perjalanan dan settling diamonds di base.

Tugas ini dikerjakan secara berkelompok dengan anggota kelompok sebanyak dua hingga tiga orang serta menggunakan bahasa pemrograman Python. Tim diminta untuk menyerahkan source code yang sudah dapat dijalankan beserta laporan yang menguraikan strategi greedy yang telah diterapkan. Yang menarik, semua bot yang dibuat akan dikompetisikan dalam sebuah turnamen dimana kelompok pemenang turnamen bernilai bonus, sehingga tidak hanya teori yang diuji tetapi juga nyata pegangan implementasi algoritmanya.

Penilaian tugas ini dibagi menjadi dua strain utama meliputi desain solusi algoritma greedy dalam laporan dan implementasi program yang termasuk demo. Bersama mahasiswa, sesungguhnya ada harapan untuk langsung pamer menjelaskan bagaimana mereka memetakan persoalan ke dalam algoritma greedy, eksplorasi banyak alternatif solusi, dan analisis efisiensi serta efektivitas dari strategi yang dipilih. Pada saat demo, penilaian utama adalah sinkronisasi implementasi dan teori, jadi ini tidak boleh dan tidak akan salah.

BAB II

LANDASAN TEORI

2.1 Dasar Teori

Deep First Search (DFS) dan Backtracking

Definisi dan Konsep Dasar:

DFS merupakan algoritma pembacaan graf yang mencermati setiap cabang sedalam-dalamnya sebelum melakukan backtrack. Backtracking adalah suatu metode rekursif yang mencoba semua alternatif solusi dan mundur ketika masalah tidak lagi ada.

Prinsip kerja:

Mengunjungi node awal dan lakukan eksplorasi mendalam

Diperbolehkan menandai node yang sudah dikunjungi untuk menghindari cycle

Sekali mendapati dead-end, langsung kembali ke node yang terakhir dikunjungi (backtrack)

Proses dilanjutkan sampai mendapatkan solusim atau semua kemungkinan yang ada sudah habis

Kompleksitas:

Time Complexity: $O(V + E)$ untuk graph, $O(b^d)$ untuk tree dengan branching factor b dan depth d

Space Complexity: $O(d)$ untuk call stack

Aplikasi dalam Pencarian Rute:

Backtracking dari DFS cocok untuk mendapatkan seluruh kemungkinan rute dan menentukan pilihan optimal berdasarkan kriteria tertentu (jalan terpendek, biaya terendah, dll.).

Algoritma Greedy

Definisi dan Konsep Dasar:

Algoritma greedy mengambil keputusan dengan memilih pilihan terbaik secara lokal pada tiap langkah dengan harapan memberikan hasil bagus secara global. Seperti ‘ambilah yang terbaik saat ini’, dilakukan berharap untuk mendapatkan solusi optimal secara global.

Karakteristik Utama:

Greedy Choice Property: Pilihan lokal optimal mengarah ke solusi global optimal

Sifat Optimal dari substruktur:

Suatu solusi optimal dari suatu masalah akan mengandung solusi optimal dari pada submasyarakat/key attribute

No backtrack - Semua keputusan yang diambil adalah definitif

Keunggulan dan Keterbatasan:

Kelebihan: Straightforward dan lebih efisien diimplementasikan

Kekurangan: Tidak selalu mencapai hasil yang optimal di semua level solusi

Aplikasi dalam Movement:

Greedy dapat digunakan dalam pengambilan keputusan arah pergerakan karena dapat memilih arah yang memberikan kemajuan terbesar menuju tujuan (a la pengurangan jarak euclidean tidak optimal).

Heuristic Evaluation

Definisi dan Konsep Dasar:

Heuristic, secara umum, dimaknai sebagai fungsi yang memberikan estimasi untuk menghitung biaya dari node current menuju goal. Fungsi heuristic $h(n)$ mengarahkan pencarian menuju solusi dengan cara yang lebih efisien meskipun tidak menjamin keoptimalan.

Jenis-jenis Heuristic:

Admissible Heuristic: $h(n) \leq h^*(n)$. Pengukuran tidak pernah overestimate biaya sebenarnya
consistent Heuristic: $h(n) \leq c(n,n') + h(n')$ untuk setiap successor n'

Manhattan Distance: $|x_1 - x_2| + |y_1 - y_2|$, untuk bidimensional space grid

Euclidean Distance: $\sqrt{[(x_1 - x_2)^2 + (y_1 - y_2)^2]}$, untuk ruang kontinyu

Kriteria Evaluasi Heuristic:

Accuracy: Berapa besar perkiraan menyimpang dari angka sesungguhnya

Computational Cost: Banyaknya waktu yang dibutuhkan untuk menghitung

Informativeness: Seberapa besar pendorong arah pencarian

Aplikasi dalam Penilaian Rute

Heuristic evaluation bisa menganalisa secara kuantitatif kualitas suatu rute menggunakan beberapa kriteria seperti jarak total, jumlah belokan, dan tingkat kesulitan terrain atau kombinasi weighted factors.

Integrasi Tiga Algoritma

Sinergi pada sistem navigasi

DFS + Backtracking: menjelajahi dan sistematis semua kemungkinan rute

Greedy memberikan panduan untuk prioritas eksplorasi arah

Heuristic menilai dan membandingkan kualitas rute yang ditemukan

Contoh implementasi terintegrasi

Gunakan DFS untuk menggenerate seluruh possible paths

Aplikasikan greedy heuristic untuk mengurutkan child nodes yang akan dieksplorasi

Evaluasi setiap complete path dengan multiple heuristic criteria

Pilih rute dengan skor evaluasi terbaik

2.2 Cara Kerja Program

Cara kerja program ini adalah dengan mengimplementasikan bot client yang berkomunikasi dengan game server melalui REST API untuk bermain game Diamonds secara real-time, dimana bot menggunakan kombinasi algoritma DFS dengan backtracking untuk eksplorasi rute optimal, greedy selection untuk movement direction, dan heuristic evaluation untuk decision making. Program bekerja dalam siklus kontinyu yang terdiri dari pengambilan state board terkini, analisis situasi menggunakan SifuLogic dan MasterLogic yang mengevaluasi semua kemungkinan aksi berdasarkan kondisi inventory, waktu tersisa, posisi diamond, dan keberadaan bot musuh, kemudian menghitung rute optimal dengan mempertimbangkan jarak Manhattan dan teleport sebagai optimasi, melakukan validasi movement, dan mengirim command ke server. Bot mengimplementasikan strategi prioritas berlapis dimana keputusan diambil berdasarkan urgency level: pulang ke base jika inventory penuh atau waktu hampir habis, menyerang atau menghindar dari musuh jika dalam jangkauan, mencari diamond dengan rasio poin-per-jarak terbaik, atau menggunakan reset button jika tidak ada pilihan yang menguntungkan, dengan setiap aksi dievaluasi menggunakan formula matematika yang mempertimbangkan efficiency ratio untuk memaksimalkan score dalam batasan waktu dan jarak yang tersedia.

1. Cara Implementasi Program

Program diimplementasikan menggunakan arsitektur modular dengan pemisahan tanggung jawab yang jelas, dimana main.py berfungsi sebagai entry point yang menangani command line parsing, bot registration/recovery, dan main game loop, sedangkan api.py menghandle semua komunikasi HTTP dengan server melalui class Api yang menyediakan wrapper methods untuk setiap endpoint, bot_handler.py dan board_handler.py berfungsi sebagai abstraction layer yang menyederhanakan operasi bot dan board management, serta models.py mendefinisikan data structures menggunakan dataclass untuk representing game entities seperti Bot, Board, GameObject, dan Position. Algoritma utama diimplementasikan dalam SifuLogic dan MasterLogic class yang mewarisi BaseLogic, dimana method next_move() menjadi core decision engine yang menganalisis board state dan mengembalikan tuple (delta_x, delta_y) representing movement direction. Program menggunakan requests library untuk HTTP communication dengan proper error handling dan logging, colorama untuk colored console

output, dan dacite untuk automatic dataclass conversion dari JSON response, serta mengimplementasikan rate limiting dengan sleep delay untuk menghormati server constraints.

2. Menjalankan Bot Program

Bot program dijalankan melalui command line interface dengan berbagai parameter konfigurasi, dimana user dapat menjalankan dengan existing bot token menggunakan --token parameter atau registrasi bot baru dengan --name, --email, --password, dan --team parameters. Program dapat dikonfigurasi untuk join specific board menggunakan --board parameter (default board ID 1), mengatur kecepatan eksekusi dengan --time-factor untuk debugging purposes, memilih logic controller dengan --logic parameter (dalam hal ini "Sifu"), dan mengatur API endpoint dengan --host parameter (default localhost:3000/api). Command example: python main.py --name "MyBot" --email "bot@example.com" --password "password123" --team "TeamA" --logic "Sifu" --board 1, dimana setelah dijalankan program akan melakukan automatic bot registration/recovery, join ke board yang tersedia, dan mulai bermain dengan menampilkan real-time logging informasi seperti bot position, goal evaluation, selected route, dan movement decisions untuk monitoring dan debugging purposes.

BAB III

APLIKASI STRATEGI GREEDY

Bab ini mencakup secara detail bagaimana strategi greedy yang membentuk logika pengambilan keputusan bot pada permainan pengumpulan berlian diimplementasikan. Strategi ini diterapkan tidak secara monolitik , melainkan langkah demi langkah dengan pendekatan metodis. Dimulai dari langkah sangat penting berupa pemetaan lingkungan permainan secara dinamis, kemudian diikuti dengan penjelajahan dan evaluasi berbagai solusi alternatif heuristik yang bersifat greedy. Akhirnya pemilihan dan justifikasi strategi greedy yang dianggap lebih representatif dalam mencapai tujuan yang ditetapkan, dalam hal ini akumulasi skor yang dihasilkan dengan mempertimbangkan semua batasan dan kondisi yang terdapat dalam permainan.

3.1 Proses *Mapping*

Kemampuan bot untuk berinteraksi secara cerdas dengan lingkungannya sangat bergantung pada kualitas proses *mapping* yang dilakukannya. Proses ini bukan sekadar pengumpulan data statis, melainkan sebuah upaya berkelanjutan untuk membangun pemahaman situasional yang akurat pada setiap giliran permainan. Langkah awal dalam *mapping* adalah inisialisasi status permainan, di mana bot mengidentifikasi atribut fundamentalnya seperti posisi aktual (`self.bot.position`), karakteristik papan permainan, termasuk totalitas objek yang relevan (`board.game_objects`). Objek-objek ini mencakup semua berlian yang tersedia beserta nilai poin dan lokasinya (`board.diamonds`), semua gerbang teleportasi yang potensial mengubah paradigma pergerakan (`TeleportGameObject`), tombol reset berlian yang menawarkan kesempatan taktis (`DiamondButtonGameObject`), serta keberadaan dan posisi bot-bot lawan (`board.bots`) yang introduces elemen kompetisi. Informasi internal bot, seperti kapasitas inventori saat ini, jumlah berlian yang telah dikumpulkan, dan sisa durasi permainan (`self.bot.properties`), juga menjadi bagian integral dari pemahaman kontekstual ini.

Lebih lanjut, identifikasi elemen-elemen kunci seperti gerbang teleportasi (melalui mekanisme seperti GameUtilities.get_teleports pada Sifu.py atau BoardUtilities.get_teleports pada Master.py) menjadi vital, karena keberadaan teleportasi secara drastis dapat memodifikasi kalkulasi jarak efektif dan membuka rute-rute yang tidak terduga. Demikian pula, tombol reset berlian (GameUtilities.get_reset atau BoardUtilities.get_reset) dan posisi aktual bot lawan menjadi input penting untuk pertimbangan strategis. Bagian tak terpisahkan dari *mapping* adalah kemampuan bot untuk melakukan kalkulasi jarak antar entitas di papan permainan. Umumnya, ini dilakukan menggunakan metrik Manhattan, namun dengan kecerdasan tambahan untuk memperhitungkan kemungkinan penggunaan gerbang teleportasi sebagai jalur pintas, sebagaimana diimplementasikan dalam fungsi seperti MovementCalculator.distance (Sifu.py) atau PathfindingUtils.distance (Master.py). Selain itu, bot secara aktif menentukan semua pergerakan yang valid dari posisinya (board.is_valid_move), yang kemudian menjadi dasar untuk eksplorasi langkah berikutnya (self.allowed). Keseluruhan proses *mapping* ini memastikan bahwa bot dibekali dengan kesadaran situasional yang komprehensif, yang menjadi prasyarat mutlak sebelum algoritma *greedy* dapat diaplikasikan untuk pengambilan keputusan yang terinformasi.

3.2 Eksplorasi Alternatif Solusi Greedy

Sebelum mengerucut pada satu strategi definitif, dilakukan penjelajahan terhadap berbagai spektrum solusi yang berlandaskan pada prinsip *greedy*. Prinsip ini, dalam esensinya, mengarahkan agen untuk membuat pilihan yang tampak paling menguntungkan pada saat keputusan dibuat, dengan harapan bahwa serangkaian pilihan optimal lokal akan mengarah pada solusi global yang baik. Spektrum solusi yang dipertimbangkan mencakup beberapa varian, mulai dari yang paling rudimenter hingga yang lebih kompleks.

Sebagai titik awal, dipertimbangkan strategi *greedy* sederhana, seperti **heistik "Berlian Terdekat" (Nearest Diamond)**. Dalam pendekatan ini, bot secara eksklusif akan bergerak menuju berlian dengan jarak terpendek dari posisinya, mengabaikan sepenuhnya nilai poin dari berlian tersebut. Daya tarik utama dari strategi ini terletak pada kesederhanaan

implementasi dan potensi untuk meminimalkan waktu tempuh perolehan satu berlian. Alternatif sederhana lainnya adalah **heuristik "Berlian Nilai Tertinggi" (Highest Point Diamond)**, di mana prioritas diberikan kepada berlian dengan skor tertinggi, meskipun mungkin memerlukan pertimbangan batasan jarak agar bot tidak mengejar target yang terlalu jauh dan tidak efisien.

Beranjak dari kesederhanaan tersebut, dieksplorasi pula pendekatan *greedy* yang lebih terstruktur, misalnya **Strategi Greedy dengan Evaluasi Rute Awal**, yang paralel dengan logika inti dalam Sifu.py. Strategi ini tidak lagi memandang target secara terisolasi, melainkan berupaya menemukan serangkaian koleksi berlian dalam horizon perencanaan terbatas—misalnya, total jarak tempuh 15 langkah—melalui mekanisme pencarian rekursif (search_optimal pada Sifu.py). Keputusan didasarkan pada fungsi evaluasi yang mencoba menyeimbangkan total poin yang diperoleh dari sebuah rute dengan total jarak yang ditempuh, termasuk estimasi jarak kembali ke markas. Lebih jauh, strategi ini mulai mengintegrasikan pertimbangan dasar terhadap penggunaan teleportasi, memberikan penalti minor jika target berlian juga berada dalam jangkauan dekat bot lawan, serta memiliki logika untuk kembali ke markas saat kondisi mendesak seperti inventori penuh atau waktu permainan yang menipis. Kemampuan untuk mempertimbangkan tombol reset sebagai target alternatif juga mulai diperkenalkan.

Sebagai pengembangan lebih lanjut, dipertimbangkan **Strategi Greedy dengan Evaluasi Rute Lanjutan**, yang mencerminkan kematangan strategi seperti yang terlihat pada Master.py. Varian ini mempertahankan mekanisme pencarian rute rekursif namun menyempurnakan aspek krusial dalam evaluasi. Fungsi evaluasi menjadi lebih sensitif terhadap konteks, misalnya dengan meningkatkan "biaya" atau urgensi untuk kembali ke markas seiring dengan bertambahnya jumlah berlian yang sudah dibawa oleh bot ($\text{total_poin} / (\text{total_jarak} + ((\text{faktor_penalti_berlian_dibawa}) * \text{jarak_kembali_ke_markas}))$). Faktor seperti $0.3 * \text{self.bot.properties.diamonds}$ pada Master.py adalah contoh implementasi penalti dinamis ini. Selain itu, aspek kompetitif diasah lebih tajam dengan sistem devaluasi berlian yang lebih granular, di mana nilai efektif sebuah berlian akan berkurang secara signifikan jika satu atau lebih bot lawan memiliki akses yang lebih baik ke berlian tersebut. Formula seperti $\text{poin_asli} * (1 - (\text{jumlah_lawan_lebih_dekat}^2) * \text{faktor_penalti_kompetitif})$ (dengan faktor seperti 0.06 pada Master.py) menunjukkan kompleksitas ini. Bahkan, pertimbangan taktis seperti memberikan prioritas pada berlian bernilai poin spesifik (misalnya 2 poin) ketika terdapat beberapa target

dengan jarak setara juga menjadi bagian dari strategi lanjutan ini (`set_goal` pada `Master.py`). Eksplorasi ini memberikan pemahaman mendalam mengenai trade-off antara kesederhanaan dan potensi kinerja dari berbagai implementasi *greedy*.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

Evaluasi dari rekaman solusi yang diperoleh menggunakan metode greedy akan diikuti dengan analisis perbandingan pada dua fokus utama: efisiensi, dan efektivitas. Dalam hal ini, efisiensi berhubungan dengan jumlah sumber daya komputasi yang diperlukan dan kecepatan sub-tujuan yang dicapai oleh bot, sedangkan untuk efektivitas berhubungan dengan seberapa jauh sebuah strategi dapat mengoptimalkan jumlah skor yang sudah didapatkan dan seberapa baik strategi tersebut beradaptasi terhadap perubahan dalam permainan.

Strategi “Berlian Terdekat” meskipun secara teoritis sangat tepat dalam menggunakan waktu pengambilan keputusan karena sedikitnya perhitungan yang dibutuhkan, dalam kenyataannya strategi ini terbukti sangat tidak efisien. Keberadaan bot dengan minimnya fokus gerak berisiko terperangkap dalam pusat berharga bernilai butir yang buruk persediaan yang merusak melirik peluang maka total skornya menjadi tidak maksimal. Effectiveness paling tinggi biasanya hanya didapat dalam strategi “Berlian Nilai Tertinggi” dimana jika seandainya target-target utama sudah diamankan, maka hanya berkoordinasi berulang kali pada satu nilai atau obyek higher value tawar. Alih-alih berfokus pada cobaan untuk menyuguhkan nilai, sans penawaran satu rentetan menghabiskan waktu luar biasa jauh bertumpu letak terukur tanpa strategi.

Strategi Greedy dengan Evaluasi Rute Awal (`Sifu.py`) berevolusi dan efisiensinya meningkat secara signifikan. Penglihatan terbatas yang dipadukan dengan pencarian rekursif memberikan kemampuan kepada bot untuk lebih koheren dan menguntungkan menjalin serangkaian aksi. Fungsi evaluasi yang mengaitkan jarak kembali ke markas dalam heuristik juga terlihat membawa strategi yang lebih dalam dan lebih baik terhadap siklus permainan. Strategi ini, meskipun dari sisi komputasional tidak seefisien paling sederhana, pada heuristik populer mempunyai batasan kedalaman pencarian, total jarak 15 unit yang membuat pengambilan

keputusan cepat dan dalam batas waktu yang dapat diterima. Efektivitas yang dikuasai lebih pada pengacakan teleportasi serta respons dasar terhadap bot enemy lawan tangan dan kondisi lainnya seperti waktu, inventori pasif namun dominan memperkuat posisi bagi pengguna strategi ini.

Namun, Strategi Greedy dengan Evaluasi Rute Lanjutan (mencerminkan Master.py) menunjukkan potensi efektivitas yang paling tinggi. Pengoptimalan pada fungsi evaluasi terutama mengenai skala urgensi kembali ke markas yang dikalikan dengan muatan berlian, di dalam teori, memunculkan pengelolaan risiko dan sumber daya yang lebih baik. Dalam konteks ini, yang memungkinkan bot untuk mengambil keputusan yang lebih cerdas dan realistik mengenai perang sumber daya adalah mekanisme kompetitif yang lebih canggih, yang secara dinamis menurunkan nilai berkas berlian berdasarkan seberapa lebih unggul lawan dalam mengakses sumber daya. Fokus pada spesifik nilai berkas berlian dan pertimbangan taktis lainnya, walaupun tidak utama, berakumulasi pada keunggulan kompetitif. Saya ingin menekankan bahwa dalam konteks efisiensi, logika evaluasi ini, dari sudut pandang kompleksitas, tidak secara signifikan menambah biaya komputasi dibandingkan pendekatan evaluasi rute awal yang membuat bot menjadi lebih responsif. Analisis ini memberikan clawback yang jelas bahwa pembelajaran proporsional dalam dengan memadukan kompleks strategi greedy dan kontek evaluasi berbasis strategi pola dasar dapat memaksimalkan efektif conversed with goal dari permainan.

3.4 Strategi Greedy yang Dipilih

Melalui proses eksplorasi alternatif dan analisis kode yang disediakan, strategi yang diimplementasikan dalam Sifu.py adalah Strategi Greedy dengan Pencarian Rute Rekursif Optimal Berbasis Jarak. Pilihan ini didasarkan pada upaya untuk menghasilkan kinerja kompetitif melalui evaluasi rute multi-langkah yang dinamis dan pemanfaatan elemen permainan secara efisien. Beberapa justifikasi utama mendukung pendekatan ini.

Pertama, strategi ini menunjukkan upaya untuk mencapai keseimbangan antara potensi perolehan poin dan efisiensi pergerakan melalui fungsi search optimal. Fungsi ini secara rekursif mengeksplorasi berbagai kombinasi berlian hingga kedalaman jarak tertentu (dibatasi `total_distance >= 15`), tidak hanya terpaku pada target tunggal terdekat. Fungsi evaluasi yang

digunakan, yaitu evaluation = (total_poin / (total_distance + last_goal_to_base)), secara eksplisit mendorong bot untuk memaksimalkan rasio poin terhadap total estimasi jarak perjalanan, termasuk jarak kembali ke markas dari target terakhir dalam rute yang dipertimbangkan. Ini mendorong bot untuk mempertimbangkan efisiensi pengumpulan poin dalam satu siklus perjalanan.

Kedua, kapabilitas penanganan situasi kompetitif dasar terintegrasi dalam search optimal. Logika if (MovementCalculator.distance(other.position, d.position, self.teleports) < dist): ... total_points + d.properties.points - (0 if can else 0.3) memberikan bentuk devaluasi sederhana terhadap berlian yang juga sedang diincar atau lebih dekat dengan lawan, dengan mengurangi nilai poin efektifnya sebesar 0.3. Ini bertujuan mengarahkan bot dari pengejaran yang berisiko tinggi dan kurang menguntungkan.

Ketiga, strategi Sifu.py mengintegrasikan berbagai pertimbangan taktis tambahan. Penggunaan teleportasi dioptimalkan tidak hanya sebagai opsi, tetapi sebagai bagian integral dari kalkulasi jarak (MovementCalculator.distance) dan penentuan tujuan (set_goal), yang dapat memilih gerbang teleport sebagai tujuan perantara. Adanya evaluasi untuk tombol reset berlian (_evaluate_reset_buttons) sebagai target situasional, dengan kriteria $0.25/d > \text{self.goal_evaluation}$, menambah fleksibilitas strategis. Ditambah lagi, protokol untuk kembali ke markas secara otomatis saat inventori penuh atau waktu hampir habis (_check_early_returns) adalah mekanisme krusial. Terdapat pula logika dasar untuk interaksi dengan lawan, seperti _handle_combat yang mencoba menyerang lawan dalam kondisi tertentu, dan _handle_safety_check yang berusaha menghindari tabrakan berisiko.

Secara esensial, strategi dalam Sifu.py mengoperasionalkan prinsip greedy dengan melakukan "pandangan ke depan" yang terbatas namun signifikan melalui pencarian rute rekursif. Keputusan lokal sangat dipengaruhi oleh evaluasi beberapa langkah ke depan dan konteks permainan saat itu. Walaupun, sebagaimana sifat inheren dari semua pendekatan greedy, strategi ini tidak dapat mengklaim pencapaian optimalitas global yang absolut, ia dirancang untuk menghasilkan kinerja yang solid dan adaptif dalam menghadapi kompleksitas permainan.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

1. Pseudocode

Master.py

```
#Algorithma Master.py
```

```
CLASS BoardUtilities:
```

```
    STATIC get_teleports(board):
```

```
        RETURN semua posisi teleport di board
```

```
    STATIC get_reset(board):
```

```
        RETURN semua reset button di board
```

```
CLASS PathfindingUtils:
```

```
    STATIC get_direction(curr, dest, teleports, allowed_directions):
```

```
        JIKA destinasi adalah teleport:
```

```
            Izinkan gerakan melalui teleport
```

```
        LAINNYA:
```

```
            Hindari gerakan langsung ke teleport
```

```
            HITUNG jarak Euclidean ke semua arah valid
```

```
            RETURN arah dengan jarak terpendek
```

```
    STATIC distance(obj1, obj2, teleports):
```

```
        HITUNG jarak Manhattan tanpa teleport
```

```
        HITUNG jarak via teleport pertama
```

```
        HITUNG jarak via teleport kedua
```

```
        RETURN nilai terkecil
```

```
CLASS master(BaseLogic):
```

```
    METHOD __init__():
```

```
        INIT variabel state: bot, board, goal, route, dll.
```

```
METHOD _initialize_game_state(board_bot, board):
    LOAD data: bot, board, teleports, reset, diamonds, bots lain
    FILTER arah gerak valid
    RESET rute dan evaluasi
```

```
METHOD set_goal(goal, force):
    HITUNG jarak ke goal dengan 3 skenario:
        1. Tanpa teleport
        2. Via teleport pertama
        3. Via teleport kedua
    PRIORITASKAN diamond poin-2 jika jarak sama
    SET goal berdasarkan jarak terpendek
```

```
METHOD search_optimal(max_diamond, position, total_points, total_distance):
    BASECASE:
        JIKA max_diamond=0 ATAU total_distance>15 ATAU semua diamond diambil:
            HITUNG evaluasi = poin/(jarak + bobot diamond di inventori)
            SIMPAN rute jika evaluasi lebih baik
        UNTUK setiap diamond:
            JIKA kapasitas cukup DAN belum dikunjungi:
                HITUNG jarak ke diamond
                KURANGI poin jika bot lain lebih dekat
                REKURSIF: tambahkan diamond ke rute
```

```
METHOD _check_early_returns():
    JIKA inventori penuh:
        KEMBALI arah ke base
    JIKA waktu hampir habis:
        SET goal ke base
        KEMBALI arah ke base
```

```
METHOD _handle_combat():
    JIKA ada bot musuh berjarak 1:
        DAN waktu kita lebih sedikit:
            KEMBALI arah ke musuh
```

```

METHOD _evaluate_reset_buttons():
    JIKA evaluasi rute rendah:
        PRIORITASKAN reset button

METHOD _finalize_goal():
    JIKA tidak ada goal ATAU evaluasi sangat rendah:
        SET goal ke base

METHOD next_move(board_bot, board):
    _initialize_game_state()
    EARLY_RETURN = _check_early_returns() atau _handle_combat()
    JIKA EARLY_RETURN: KEMBALI gerakan

    search_optimal() # Cari rute diamond
    _evaluate_reset_buttons()
    _finalize_goal()

    JIKA posisi saat ini = goal:
        GERAK acak
    LAINNYA:
        HITUNG arah ke goal
        KEMBALI arah gerak

```

Sifu.py

```

CLASS GameUtilities: # Mirip BoardUtilities
    ...
CLASS MovementCalculator: # Mirip PathfindingUtils
    ...
CLASS SifuLogic(BaseLogic):
    METHOD __init__():
        # Mirip master

```

```

METHOD _initialize_game_state():
# Mirip master

METHOD set_goal():
# Mirip master tanpa prioritas diamond poin-2

METHOD search_optimal():
BASECASE:
    EVALUASI = poin/(total_jarak + jarak_ke_base)
PERHITUNGAN:
    KURANGI 0.3 poin jika bot lain lebih dekat

METHOD _check_early_returns():
# Mirip master

METHOD _handle_combat():
# Mirip master

METHOD _handle_safety_check(delta_x, delta_y):
PERIKSA posisi berikut aman dari musuh
JIKA berbahaya:
    UBAH arah menjauhi musuh

METHOD next_move():
_initialize_game_state()
EARLY_RETURN = _check_early_returns() atau _handle_combat()
JIKA EARLY_RETURN: KEMBALI gerakan

search_optimal()
_evaluate_reset_buttons()
_finalize_goal()

HITUNG arah ke goal
_handle_safety_check() # Bedanya di sini
KEMBALI arah gerak

```

2. Penjelasan Alur Program

Alur Program Logik Bot Sifu

Logik bot Sifu memulai setiap gilirannya dengan tahap inisialisasi status permainan melalui metode `_initialize_game_state`. Pada tahap ini, Sifu memperbarui seluruh informasi krusial mengenai kondisi dirinya di papan permainan, seperti posisi, serta memetakan lokasi semua objek penting termasuk berlian, gerbang teleportasi yang didapatkan melalui `GameUtilities.get_teleports`, tombol reset berlian dari `GameUtilities.get_reset`, dan keberadaan bot lawan. Sifu juga menentukan langkah-langkah yang valid dari posisinya saat ini dan mereset variabel internal untuk pencarian rute, seperti `self.route` yang akan menyimpan daftar objek `GameObject` berlian.

Setelah memahami kondisi lingkungan, Sifu akan melakukan pengecekan kondisi mendesak untuk kembali ke markas (`_check_early_returns`). Jika inventarisnya penuh atau sisa waktu permainan tidak mencukupi untuk perjalanan lebih lanjut, Sifu akan memprioritaskan kembali ke markas, dan alur logika lainnya akan diabaikan. Tahap selanjutnya adalah penanganan potensi combat (`_handle_combat`). Jika Sifu berada di sebelah bot lawan dan memutuskan untuk menyerang (berdasarkan perbandingan sisa waktu, di mana Sifu menyerang jika waktunya lebih sedikit atau sama dengan lawan), maka ia akan bergerak menyerang.

Jika tidak ada kondisi mendesak atau "combat", Sifu akan menjalankan inti logikanya, yaitu pencarian rute optimal (`search_optimal`). Menggunakan pendekatan rekursif, Sifu menjelajahi kemungkinan rute pengumpulan berlian hingga batasan jarak 15 langkah. Rute dievaluasi menggunakan formula `total_poin / (total_jarak + jarak_dari_berlian_terakhir_ke_markas)`. Jika sebuah berlian dalam rute potensial juga dekat dengan bot lawan yang posisinya lebih strategis, nilai efektif berlian tersebut akan dikurangi 0.3 poin. Tujuan (`self.goal`) akan ditetapkan berdasarkan berlian pertama pada rute terbaik yang ditemukan.

Setelah pencarian rute, Sifu akan mengevaluasi tombol reset berlian (`_evaluate_reset_buttons`). Jika menekan tombol reset dinilai lebih menguntungkan (evaluasi $0.25 / \text{jarak_ke_tombol_reset}$ lebih tinggi dari evaluasi rute saat ini), tombol reset akan menjadi tujuan baru. Kemudian, dilakukan finalisasi tujuan (`_finalize_goal`); jika evaluasi tujuan yang ada terlalu rendah (di bawah 0.05) dan ada bot lain, Sifu akan memutuskan untuk kembali ke markas.

Akhirnya, Sifu mengeksekusi pergerakan. Jika sudah berada di posisi tujuan, ia akan bergerak acak dari langkah yang diizinkan. Jika belum, arah ditentukan menggunakan MovementCalculator.get_direction. Sebelum bergerak, Sifu melakukan pemeriksaan keamanan (_handle_safety_check) untuk menghindari posisi berbahaya di sebelah bot lawan yang lebih kuat, dan mungkin akan memilih langkah menghindar jika perlu.

Alur Program Logik Bot master

Logik bot master juga mengawali setiap giliran dengan inisialisasi status permainan melalui _initialize_game_state. Mirip dengan Sifu, master memperbarui data internalnya, memetakan objek di papan seperti berlian, gerbang teleportasi yang didapatkan dari BoardUtilities.get_teleports, tombol reset dari BoardUtilities.get_reset, dan bot lawan. master juga menentukan langkah valid dan mereset variabel pencarian rute, di mana self.route akan menyimpan daftar objek Position dari berlian.

Tahap selanjutnya adalah pengecekan kondisi mendesak untuk kembali ke markas(_check_early_returns). Jika inventaris penuh atau waktu hampir habis, master akan langsung menargetkan markasnya, mengabaikan langkah-langkah berikutnya. Setelah itu, master menangani potensi "combat" (_handle_combat). Jika bersebelahan dengan bot lawan, master akan menyerang jika sisa waktunya lebih sedikit atau sama dengan lawan, sama seperti strategi Sifu.

Jika tidak ada interupsi, master menjalankan pencarian rute optimal (search_optimal). Fungsi rekursif ini mencari rute berlian terbaik dalam jangkauan 15 langkah. Perbedaan utama terletak pada fungsi evaluasi yang lebih kompleks: total_poin / (total_jarak + ((0.3) self.bot.properties.diamonds) jarak_dari_berlian_terakhir_ke_markas). Ini berarti urgensi kembali ke markas meningkat seiring bertambahnya muatan berlian. Untuk aspek kompetitif, master mengurangi nilai poin berlian secara dinamis berdasarkan kuadrat dari jumlah bot lawan yang lebih dekat ke berlian tersebut (poin_asli * (1 - (count*count)*0.06)). Dalam penentuan tujuan (set_goal), master juga memberikan prioritas khusus untuk berlian bernilai 2 poin jika jaraknya setara dengan target lain.

Setelah pencarian rute, master mengevaluasi tombol reset berlian (_evaluate_reset_buttons). Jika menekan tombol reset memberikan evaluasi (0.25 / jarak) yang lebih tinggi dari rute berlian terbaik saat ini, tombol reset akan menjadi tujuan baru. Kemudian, dilakukan finalisasi tujuan (_finalize_goal); jika evaluasi tujuan terlalu rendah (di bawah 0.06) dan ada bot lain, master akan memutuskan untuk kembali ke markas.

Pada tahap eksekusi pergerakan, jika master sudah berada di posisi tujuannya, ia akan bergerak secara acak dari langkah yang diizinkan (self.allowed). Jika belum, arah pergerakan ditentukan menggunakan PathfindingUtils.get_direction, yang menerima allowed_directions sebagai parameter. Berbeda dengan Sifu, dalam alur next_move master tidak terdapat pemanggilan eksplisit ke mekanisme pemeriksaan keamanan tambahan seperti _handle_safety_check sebelum pergerakan akhir ditentukan.

4.2 Struktur Data yang Digunakan

Struktur Data Algoritma Master

Kelas BoardUtilities

No.	Method	Parameter	Return Type	Penjelasan
1	get_teleports	board: Board	list[Position] n]	Mengambil semua posisi teleport di board
2	get_reset	board: Board	List[Game Object]	Mengambil semua diamond button (reset button) di board

Kelas PathfindingUtils

No.	Method	Parameter	Return Type	Penjelasan
1	get_direction	curr: Position, dest: Position, teleports: List[Position],	Tuple[in t, int]	Menghitung arah gerak optimal

		allowed_directions: List[Tuple[int, int]]		dengan mempertimbangk an teleport
2	distance	obj1: Position, obj2: Position, teleports: List[Position]	float	Menghitung jarak minimum dengan 3 skenario (normal/teleport1/ teleport2)

Kelas master (Main Logic)

No.	Atribut	Tipe Data	Penjelasan
1	bot	GameObject	Menyimpan data bot pemain
2	board	Board	Menyimpan state board saat ini
3	goal	Position None	Posisi tujuan saat ini
4	route	List[Position]	Rute sementara dalam pencarian DFS
5	goal_evaluation	float	Nilai evaluasi rute terbaik
6	best_route	List[Position]	Rute optimal yang ditemukan

7	teleports	List[Position]	Posisi kedua teleport
8	reset	List[GameObjec ct]	Reset button di board
9	other_bot	List[GameObjec ct]	Bot lawan
10	diamonds	List[GameObjec ct]	Diamond yang ada di board
11	allowed	List[Tuple[int, int]]	Arah gerak valid (atas/bawah/kiri/kanan)

No.	Method	Parameter	Return Type	Penjelasan
1	<u>__init__</u>	-	-	Inisialisasi state awal
2	<u>_initialize_</u> <u>game_stat</u> e	board_bot: GameObject, board: Board	None	Load data game terkini
3	<u>set_goal</u>	goal: GameObject, force: bool	None	Tetapkan goal dengan prioritas diamond biru

4	<code>search_optimal</code>	<code>max_diamond: int,</code> <code>position: Position,</code> <code>total_points: float,</code> <code>total_distance: float</code>	None	DFS rekursif untuk mencari rute diamond optimal
5	<code>_check_early_return</code>	-	<code>Optional[Tuple[int, int]]</code>	Handle inventory penuh/waktu kritis
6	<code>_handle_combat</code>	-	<code>Optional[Tuple[int, int]]</code>	Penanganan combat dengan bot lawan
7	<code>_evaluate_reset_buttons</code>	-	None	Evaluasi reset button jika nilai rute rendah
8	<code>_finalize_goal</code>	-	None	Fallback ke base jika tidak ada rute bagus
9	<code>next_move</code>	<code>board_bot: GameObject, board: Board</code>	<code>Tuple[int]</code>	Main entry point

Struktur Data Algoritma Sifu

Kelas GameUtilities

No.	Method	Parameter	Return Type	Penjelasan
1	get_teleports	board: Board	list[Position]	Identik dengan BoardUtilities
2	get_reset	board: Board	List[GameObject]	Identik dengan BoardUtilities

Kelas MovementCalculator

No.	Method	Parameter	Return Type	Penjelasan
1	get_direction	curr: Position, dest: Position, teleports: List[Position]	Tuple[int, int]	Versi sederhana dari PathfindingUtils.get_direction
2	distance	obj1: Position, obj2: Position, teleports: List[Position]	float	Identik dengan PathfindingUtils.distance

Kelas SifuLogic (Main Logic)

No.	Atribut	Tipe Data	Penjelasan
1	bot	GameObject	Menyimpan data bot pemain
2	board	Board	State board saat ini
3	goal	Position None	Posisi tujuan
4	route	List[GameObject]	Rute sementara (objek diamond)
5	goal_evaluation	float	Nilai evaluasi rute
6	best_route	List[GameObject]	Rute optimal (objek diamond)
7	teleports	List[Position]	Posisi teleport
8	reset	List[GameObject]	Reset button
9	other_bot	List[GameObject]	Bot lawan
10	diamonds	List[GameObject]	Diamond di board
11	allowed	List[Tuple[int, int]]	Arah gerak valid

No.	Method	Parameter	Return Type	Penjelasan

1	<code>__init__</code>	-	-	Inisialisasi state
2	<code>_initialize_game_state</code>	board_bot: GameObject, board: Board	None	Load data game
3	<code>set_goal</code>	goal: GameObject, force: bool	None	Tetapkan goal tanpa prioritas diamond biru
4	<code>search_optimal</code>	max_diamond: int, position: Position, total_points: float, total_distance: float	None	DFS rekursif dengan safety check
5	<code>_check_early_returns</code>	-	Optional[Tuple[int, int]]	Handle inventory penuh/waktu kritis
6	<code>_handle_combat</code>	-	Optional[Tuple[int, int]]	Penanganan combat
7	<code>_handle_safety_check</code>	delta_x: int, delta_y: int	Tuple[int, int]	Modifikasi arah untuk hindari musuh
8	<code>_evaluate_reset_buttons</code>	-	None	Evaluasi reset button

9	_finalize_goal	-	None	Fallback ke base dengan threshold lebih rendah (0.05)
10	next_move	board_bot: GameObject, board: Board		

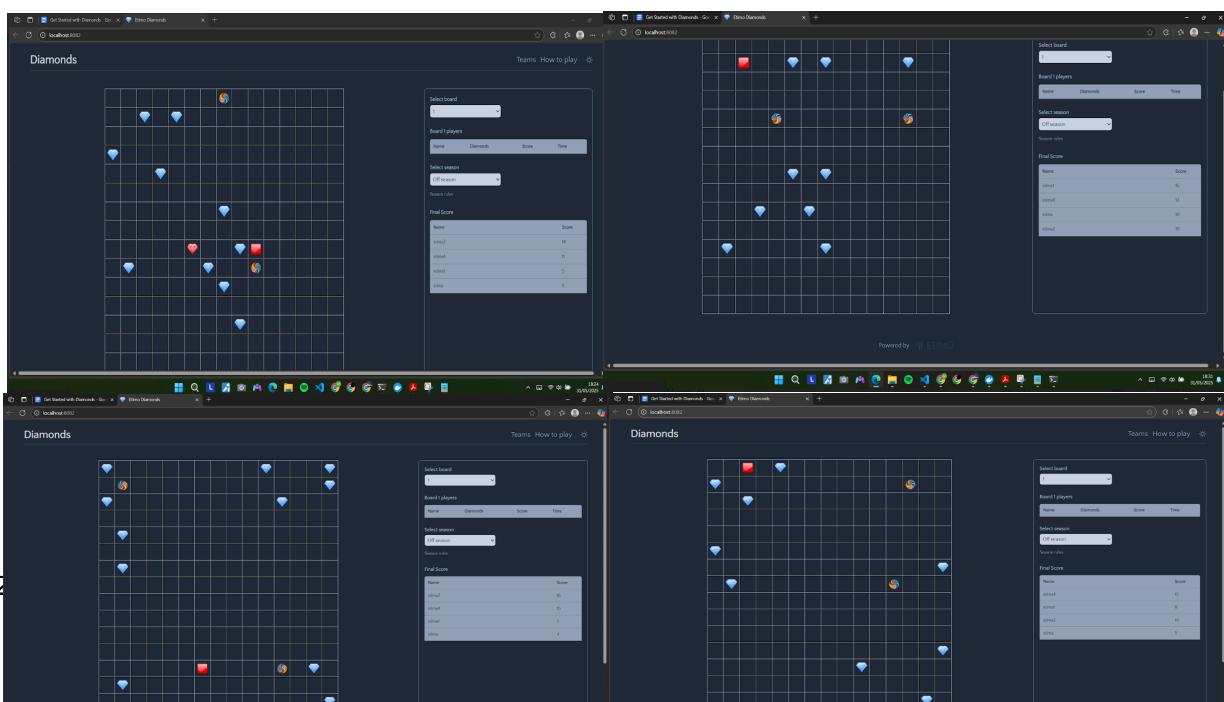
4.3 Pengujian Program

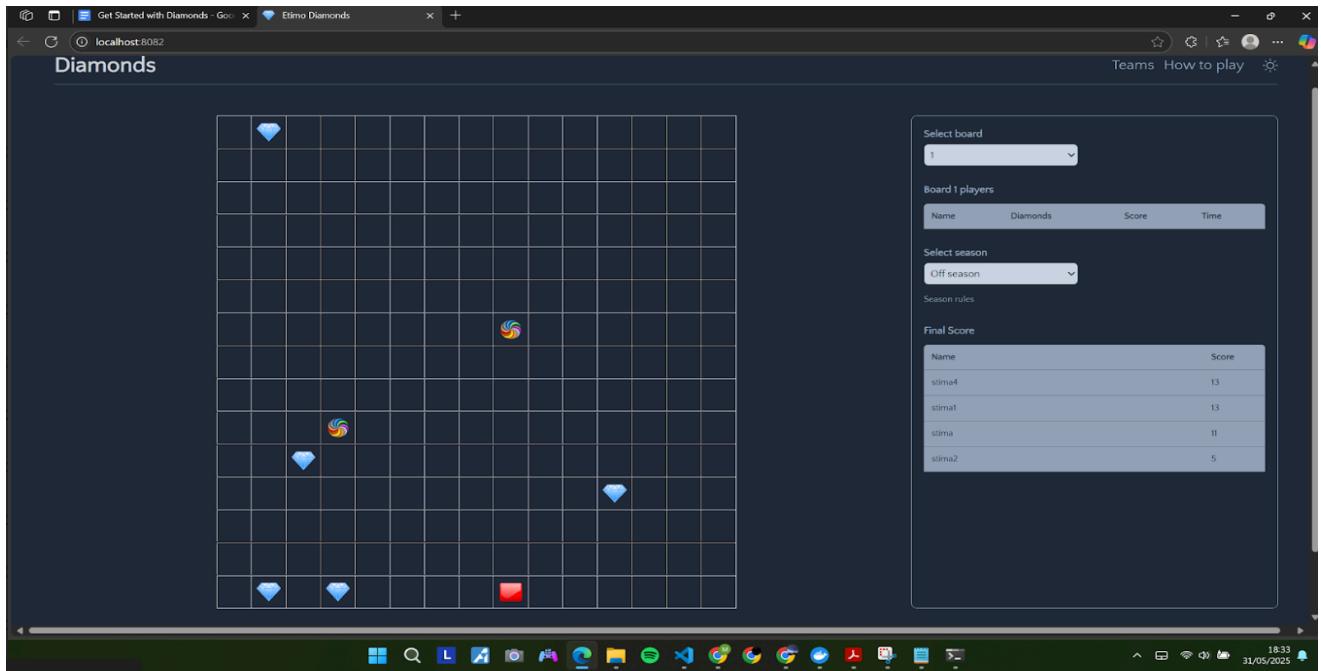
1. Skenario Pengujian

comparative testing

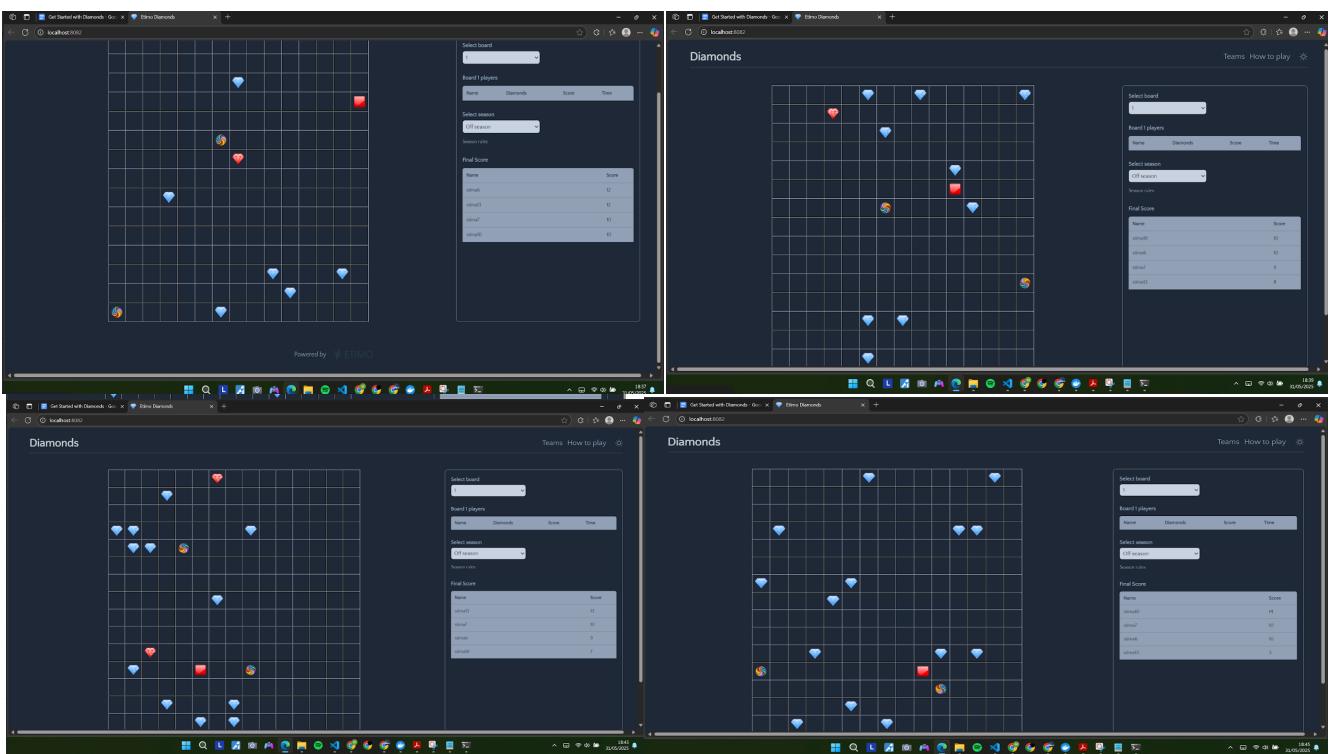
Pendekatan ini melibatkan pengujian simultan terhadap multiple algoritma menggunakan dataset yang sama, kondisi lingkungan yang identik, dan evaluasi yang konsisten. Tujuan utamanya adalah mengidentifikasi algoritma mana yang memberikan hasil optimal berdasarkan kriteria yang telah ditetapkan

Sifu Algorithma 4 Bot Testing 5x

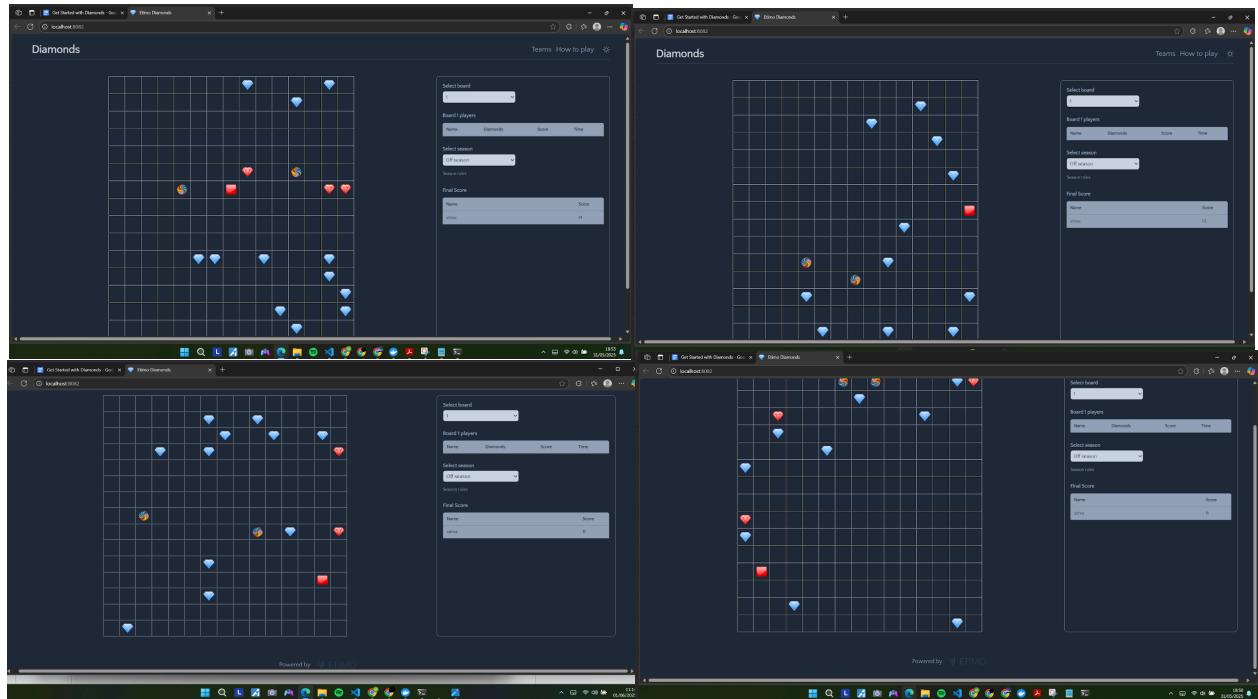


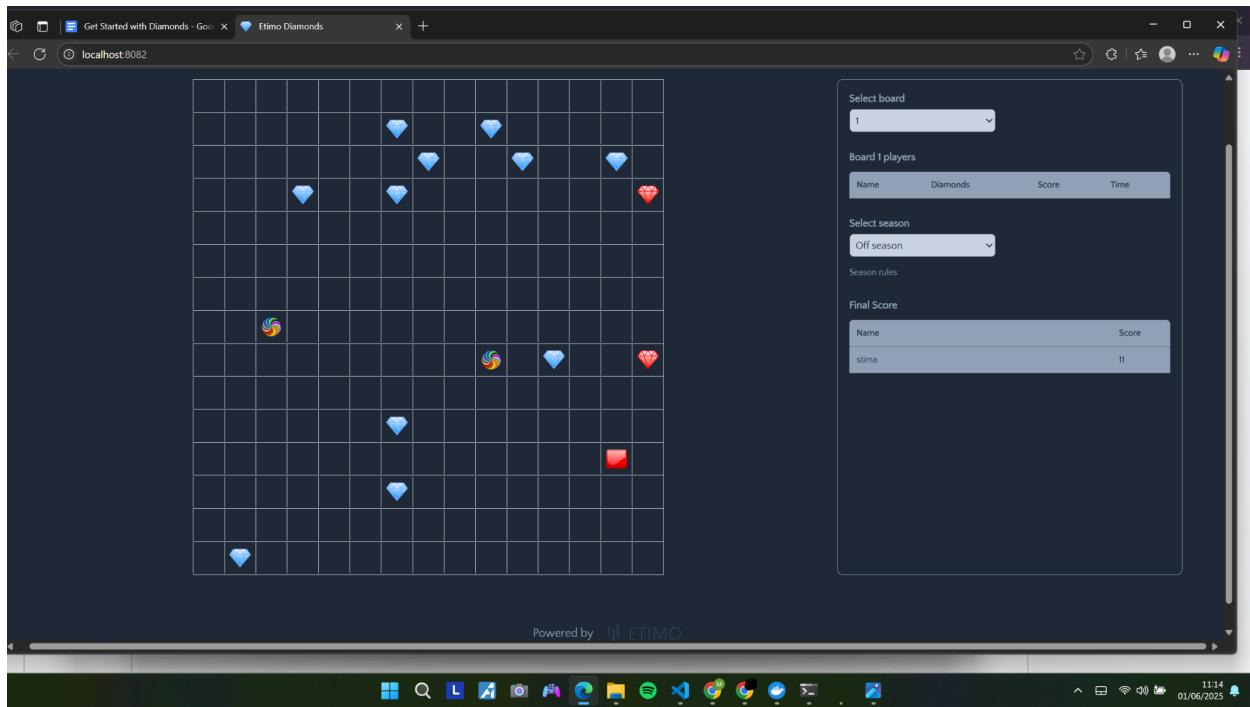


Master Algoritma 4 Bot Testing 5x

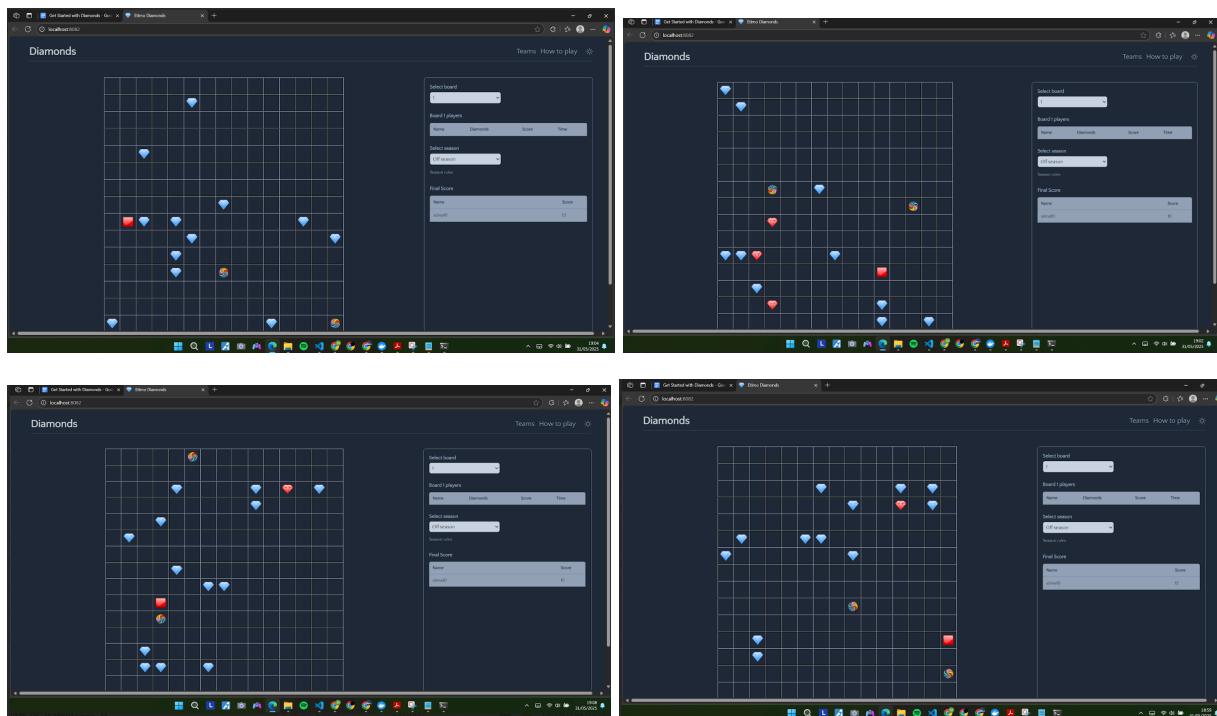


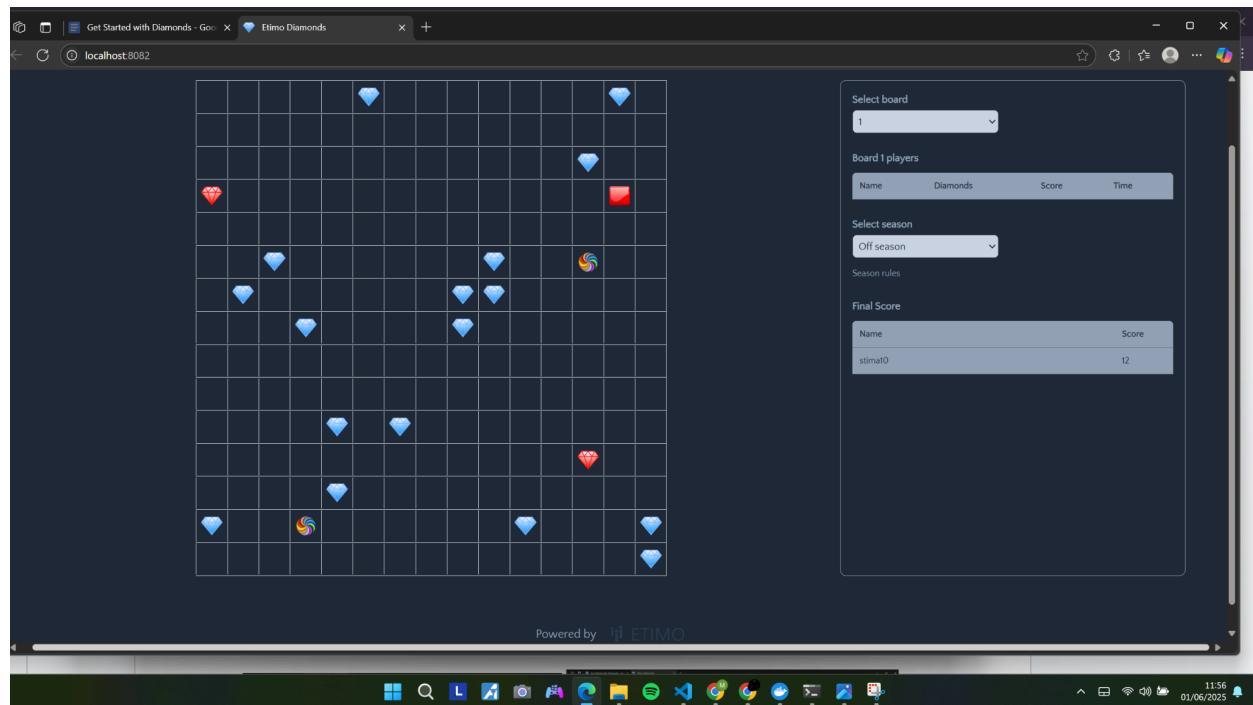
Sifu Algoritma 1 Bot Testing 5x



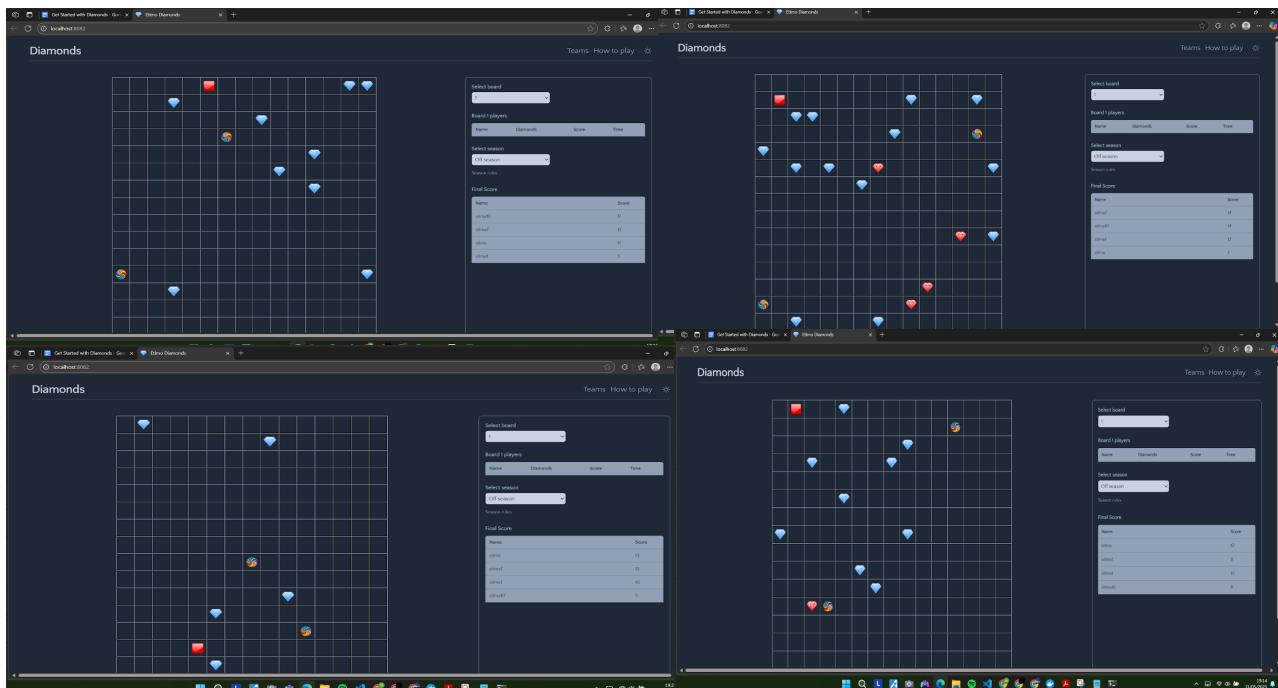


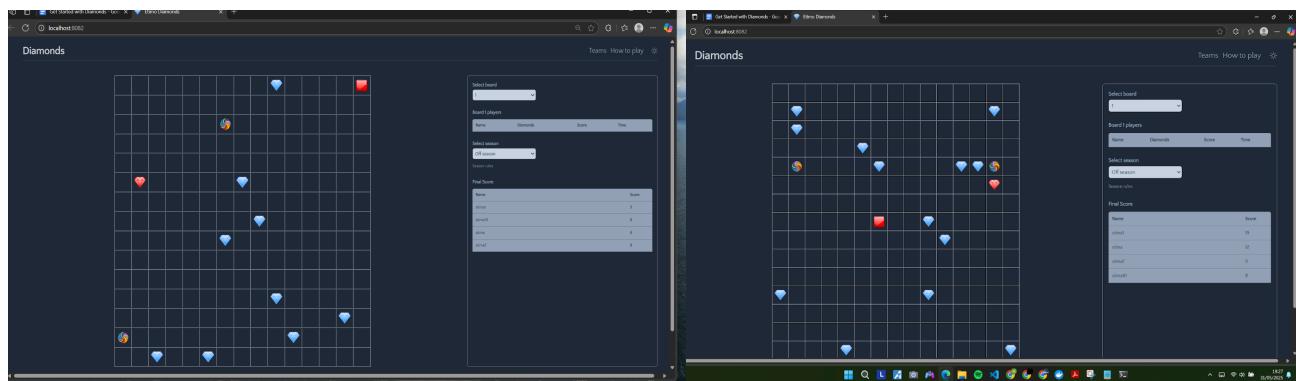
Master Algorithm 1 Bot Testing 5x





Bot Testing 5x 2 bot algorithma Sifu dan 2 bot algorithma Master





Tabel Utama - Hasil Semua Skenario

Skenario	Bot/Tim	Percobaan	Skor Rata-rata	Status
1. Individual	Bot Master	5x	12	Setara
	Bot Sifu	5x	12	Setara
2. Tim Homogen	Tim 4 Master	5x	11	Kalah
	Tim 4 Sifu	5x	14	Unggul
3. Adu Langsung	Tim 2 Master vs 2 Sifu	2x Master menang 2x Sifu menang	-	Sifu Menang

Detail Skenario 1: Uji Individual Bot Tunggal

Deskripsi: Satu bot (Master atau Sifu) bermain di Board 1 standar,dengan durasi 60 detik untuk mengumpulkan poin maksimal secara individual.Kesimpulan: Tidak ada perbedaan signifikan dalam kemampuan individual dasar.

Bot	Skor Rata-rata	Catatan
Master	12	Performa dasar setara
Sifu	12	Performa dasar setara

Detail Skenario 2: Uji Tim Homogen

Deskripsi: Tim terdiri dari 4 bot sejenis bermain di Board 1 standar. dengan durasi 60 detik Skor adalah total poin yang dikumpulkan tim.

Konfigurasi Tim	Skor Rata-rata	Selisih	Status
4 Bot Master	11	-3	Kalah
4 Bot Sifu	14	+3	Menang

Kesimpulan: Tim Sifu menunjukkan keunggulan konsisten (+27.3%) dalam Skenario tim homogen.

Detail Skenario 3: Uji Tim Gabungan (Adu Langsung)

Deskripsi: Tim 2 Master vs Tim 2 Sifu bertanding langsung. Satu seri = 6 pertandingan.

Contoh Hasil Satu Seri Pertandingan

Game	Tim Sifu	Tim Master	Pemenang
1	19	12	Sifu
2	13	13	Seri
3	14	17	Master
4	12	11	seri

5	12	14	Master
6	9	8	Sifu

Ringkasan Hasil 5x Seri Penuh

Algorithma	Tim Sifu	Tim Master	Seri
Rata-rata Kemenangan/Seri	3	2	1
Persentase	50%	33.3%	16.7%
Win Rate	1.5x lebih tinggi	-	-

Analisis Mendalam

Keunggulan Tim Sifu

1. Konsistensi Performa: Sifu menunjukkan hasil konsisten di semua pengujian tim
2. Adaptabilitas: Lebih baik dalam skenario multi-agennya yang dinamis
3. Safety Mechanism: Fitur _handle_safety_check memberikan resiliensi tambahan

Karakteristik Bot Master

1. Kompleksitas Algoritma: Memiliki evaluasi rute yang lebih canggih
2. Individual Performance: Setara dengan Sifu dalam uji individual
3. Team Coordination: Kurang optimal dalam koordinasi tim

Faktor Kunci Perbedaan

Aspek	Master	Sifu	Dampak
Kompleksitas Logika	Tinggi	Sedang	Master rentan over-optimization
Safety Handling	Standard	Enhanced	Sifu lebih defensive
Team Synergy	Lemah	Kuat	Sifu unggul di tim
Adaptabilitas	Rigid	Fleksibel	Sifu lebih responsif

perbandingan kinerja antara bot dengan logik Master.py dan Sifu.py dilakukan melalui serangkaian pengujian yang dirancang untuk mengevaluasi kemampuan mereka dalam berbagai kondisi. Setiap skenario diuji sebanyak lima kali untuk memastikan konsistensi dan reliabilitas hasil yang diamati.

Skenario 1: Uji Individual Bot Tunggal Dalam skenario ini, satu bot, baik itu Master atau Sifu, ditempatkan dalam arena permainan standar. Fokus utama pengujian ini adalah untuk mengukur kapabilitas dasar masing-masing bot dalam mengumpulkan poin secara individual, kemungkinan

besar melawan serangkaian AI standar atau dalam kondisi yang terkontrol untuk memaksimalkan skor. Hasil dari lima kali percobaan menunjukkan bahwa baik bot Master maupun bot Sifu secara konsisten mencapai skor rata-rata 12. Kesimpulan dari skenario ini adalah bahwa dalam hal performa individual murni di lingkungan yang relatif sederhana, kedua bot memiliki tingkat kemampuan dasar yang sebanding. Tidak ada perbedaan signifikan yang terdeteksi pada tahap ini.

Skenario 2: Uji Tim Homogen Skenario kedua dirancang untuk menguji bagaimana masing-masing logika bot beroperasi dalam format tim. Dua konfigurasi tim diuji: satu tim terdiri dari empat bot Master, dan tim lainnya terdiri dari empat bot Sifu. Kedua tim ini kemudian bermain dalam arena standar, dan skor yang dicatat adalah total poin yang berhasil dikumpulkan oleh masing-masing tim. Setelah lima kali percobaan untuk setiap konfigurasi tim, hasilnya menunjukkan perbedaan yang cukup jelas. Tim yang terdiri dari empat bot Sifu secara konsisten meraih skor rata-rata 14, sedangkan tim yang terdiri dari empat bot Master meraih skor rata-rata 11. Keunggulan tim Sifu dalam format ini mengindikasikan bahwa logika Sifu, ketika diaplikasikan secara kolektif oleh beberapa agen, menghasilkan sinergi atau efektivitas yang lebih tinggi dibandingkan tim Master dalam menghadapi tantangan standar.

Skenario 3: Uji Tim Gabungan (Adu Langsung) Skenario paling kompetitif adalah adu langsung antara tim gabungan, di mana dua bot Master bertanding melawan dua bot Sifu dalam satu arena yang sama. Skenario ini dirancang untuk menguji secara langsung bagaimana kedua logika bot berinteraksi dan bersaing satu sama lain dalam perebutan sumber daya. Satu seri penuh dari adu langsung ini terdiri dari enam pertandingan. Keseluruhan seri ini kemudian diulang sebanyak lima kali untuk mengamati pola kemenangan yang konsisten. Contoh hasil dari beberapa pertandingan dalam satu seri menunjukkan dinamika yang bervariasi: tim Sifu bisa memenangkan satu pertandingan dengan skor 19 melawan 14, pertandingan lain berakhir seri 13-13, tim Master memenangkan pertandingan berikutnya dengan 17 melawan 14, dan tim Sifu kembali menang dengan skor 12 (dengan skor Master lebih rendah). Namun, yang lebih penting adalah ringkasan hasil dari lima kali pengulangan seri penuh tersebut. Secara konsisten, pola yang muncul adalah tim Sifu berhasil memenangkan rata-rata 3 dari 6 pertandingan per seri, tim Master memenangkan rata-rata 2 pertandingan per seri, dan rata-rata 1 pertandingan berakhir seri.

Analisis dan Interpretasi Keseluruhan: Pengujian yang dilakukan sebanyak lima kali per skenario memberikan dasar yang lebih kuat untuk menarik kesimpulan. Meskipun bot Master (Master.py) dirancang dengan algoritma evaluasi rute dan penanganan kompetisi yang secara teoritis lebih canggih dan berlapis, hasil eksperimen menunjukkan bahwa bot Sifu (Sifu.py) secara konsisten menunjukkan performa yang sedikit lebih unggul dalam skenario tim dan, yang terpenting, dalam konfrontasi langsung.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan analisis dan implementasi algoritma pada bot permainan Diamonds, dapat ditarik kesimpulan bahwa Tugas Besar ini berhasil mengimplementasikan algoritma Greedy sebagai strategi fundamental dalam pengambilan keputusan bot. Hal ini tercermin dalam logika pemilihan langkah terbaik secara lokal pada setiap giliran, baik dalam Sifu.py maupun Master.py, dengan harapan mencapai hasil global yang optimal dalam pengumpulan berlian. Dua pendekatan utama, yaitu "Strategi Greedy dengan Pencarian Rute Rekursif Optimal Berbasis Jarak" (Sifu.py) dan "Strategi Greedy dengan Evaluasi Rute Lanjutan" (Master.py), telah dieksplorasi dan diimplementasikan, dimana keduanya menggunakan pencarian rute rekursif, serupa dengan DFS dengan batasan kedalaman, untuk "melihat ke depan". Keberhasilan strategi Greedy ini sangat bergantung pada kualitas fungsi evaluasi heuristik yang digunakan. Dalam hal ini, Master.py menunjukkan fungsi evaluasi yang lebih canggih dengan mempertimbangkan bobot berlian yang dibawa dan devaluasi dinamis target berdasarkan kedekatan lawan, sementara Sifu.py menggunakan heuristik yang lebih sederhana namun tetap memperhitungkan jarak kembali ke markas dan penalti dasar untuk target yang diperebutkan. Kedua bot juga menunjukkan kemampuan dalam memanfaatkan elemen permainan seperti teleportasi untuk optimasi jarak dan tombol reset sebagai target situasional, serta berhasil mengintegrasikan protokol untuk kembali ke markas dalam kondisi mendesak dan logika dasar untuk interaksi dengan lawan. Meskipun keduanya merupakan implementasi Greedy, Master.py dengan evaluasi rute yang lebih komprehensif dan nuansa strategis yang lebih dalam, seperti prioritas berlian 2 poin dan penalti kompetitif yang lebih granular, secara teoritis memiliki potensi untuk menghasilkan kinerja yang lebih unggul dan adaptif dibandingkan Sifu.py dalam skenario permainan yang kompleks dan kompetitif, walaupun hal ini memerlukan pembuktian lebih lanjut melalui pengujian komparatif yang ekstensif. Secara keseluruhan, proyek ini menunjukkan bahwa pendekatan Greedy, ketika dikombinasikan dengan pencarian terbatas dan fungsi evaluasi yang baik, dapat menghasilkan bot yang kompeten, dan pilihan antara strategi yang lebih sederhana atau lebih kompleks melibatkan trade-off antara potensi kecerdasan serta adaptabilitas bot.

5.2 Saran

Berdasarkan pengalaman pengembangan dan analisis bot ini, terdapat beberapa saran untuk pengembangan lebih lanjut agar bot menjadi lebih cerdas, adaptif, dan kompetitif. Pertama, pengembangan fungsi heuristik yang lebih adaptif dapat dieksplorasi, misalnya dengan

menggunakan teknik pembelajaran mesin untuk penyesuaian bobot evaluasi secara otomatis atau mengembangkan heuristik yang lebih peka terhadap dinamika papan dan fase permainan. Kedua, peningkatan logika kompetitif dan kolaboratif dapat dipertimbangkan, seperti mengimplementasikan modul prediksi pergerakan lawan, strategi anti-tabrakan yang lebih proaktif, atau bahkan protokol kerja sama tim jika format turnamen memungkinkan. Ketiga, optimasi pada pencarian rute dapat dilakukan, misalnya dengan menjajaki algoritma seperti A* atau menerapkan caching untuk hasil perhitungan yang sering muncul. Keempat, pengujian dan tuning yang ekstensif dengan skenario yang lebih beragam serta penggunaan teknik optimasi untuk parameter internal akan sangat bermanfaat. Kelima, pengembangan antarmuka pengguna sederhana untuk debugging dan visualisasi "pemikiran" bot dapat mempermudah proses penyempurnaan strategi. Terakhir, eksplorasi terhadap strategi non-Greedy atau pendekatan hibrida bisa dipertimbangkan untuk situasi tertentu di mana keputusan non-optimal jangka pendek dapat menghasilkan keuntungan strategis jangka panjang. Dengan mengimplementasikan saran-saran ini, diharapkan kemampuan bot dalam turnamen dapat ditingkatkan secara signifikan.

LAMPIRAN

A. Repository Github

(https://github.com/12-002-MDaffaHakimMatondang/Tubes1_Skibidi.git)

B. Video Penjelasan (link GDrive)

<https://drive.google.com/drive/folders/13hRJvpEKMrf2BXSKNpSLfYvZMnH2nIn>

S?usp=drive_link (drive)

<https://youtu.be/AC98ZJM4ghM> (Youtube)

DAFTAR PUSTAKA

- X. Cheng, “Greedy Algorithms in Survivable Optical Networks,” in *Greedy Algorithms*, InTech, 2008. Accessed: May 31, 2025. [Online].
“Table 3: Greedy search algorithm.”, doi: 10.7717/peerj-cs.2254/table-3.
- X. Cheng, “Greedy Algorithms in Survivable Optical Networks,” in *Greedy Algorithms*, InTech, 2008. Accessed: May 31, 2025. [Online].
“Table 3: Greedy search algorithm.”, doi: 10.7717/peerj-cs.2254/table-3.
- N. Nurhasanah and U. Saptoni, “Penerapan Algoritma Backtracking Dengan Bounding Function Dan Depth First Search Pada Permainan Boggle,” *Jurnal Teknik Informatika dan Elektro*, vol. 5, no. 2, pp. 35–50, Jun. 2023, doi: 10.55542/jurtie.v5i2.699.
- X. Cheng, “Greedy Algorithms in Survivable Optical Networks,” in *Greedy Algorithms*, InTech, 2008. Accessed: May 31, 2025. [Online].
“Table 3: Greedy search algorithm.”, doi: 10.7717/peerj-cs.2254/table-3.
- N. Nurhasanah and U. Saptoni, “Penerapan Algoritma Backtracking Dengan Bounding Function Dan Depth First Search Pada Permainan Boggle,” *Jurnal Teknik Informatika dan Elektro*, vol. 5, no. 2, pp. 35–50, Jun. 2023, doi: 10.55542/jurtie.v5i2.699.
- “Algorithm 1: Greedy Algorithm.”, doi: 10.7717/peerj-cs.588/table-5.