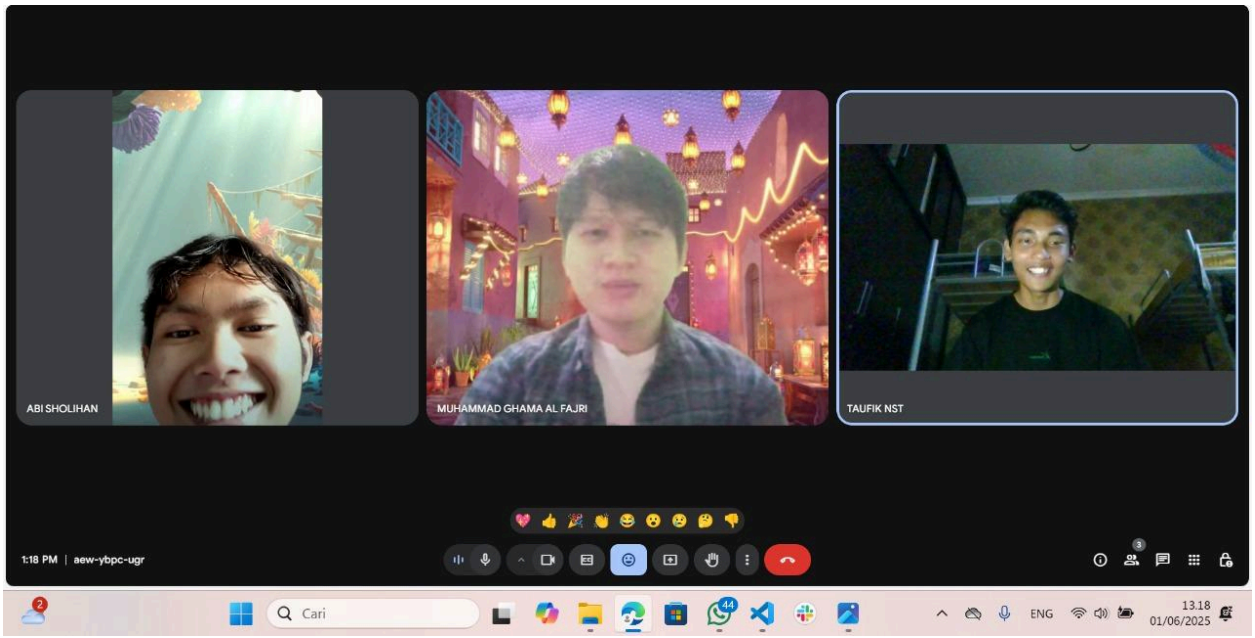
LAPORAN TUGAS BESAR MATA KULIAH
STRATEGI ALGORITMA



Oleh:

Abi Sholihan	123140192
Muhammad Ghama Al-Fajri	123140182
Taufik Hidayat NST	123140188

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

INSTITUT TEKNOLOGI SUMATERA

2025

DAFTAR ISI

BAB I
DESKRIPSI TUGAS

I.I Program Diamonds

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.

Program permainan Diamonds terdiri dari:

1. *Game enjine,* yang secara umum berisi:
    a. Kode backend permainan, yang berisi logic permainan secara keseluruhan serta API yang disediakan untuk berkomunikasi dengan frontend dan program bot
    b. Kode frontend permainan, yang berfungsi untuk memvisualisasikan permainan
2. *Bot starter pack,* yang secara umum berisi:
    a. Program untuk memanggil API yang tersedia pada backend
    b. Program bot logic (bagian ini yang akan kalian implementasikan dengan algoritma greedy untuk bot kelompok kalian)
    c. Program utama (main) dan utilitas lainnya

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.

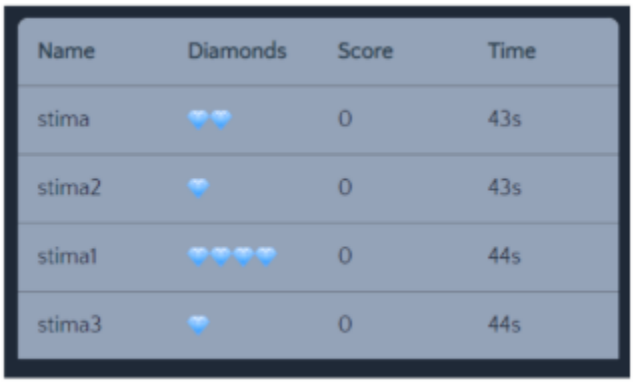
3. Teleporters

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.

1.2 Ketentuan Tugas Besar

Berikut ini adalah ketentuan-ketentuan dalam pengerjaan tugas besar mata kuliah strategi algoritma:

1. Tugas dikerjakan secara berkelompok dengan anggota minimal 2 orang dan maksimal 3 orang.
2. Mahasiswa diminta untuk membuat program sederhana dalam bahasa pemrograman python yang mengimplementasikan algoritma Greedy pada bot permainan Diamonds dengan tujuan memenangkan permainan.
3. Strategi greedy yang diimplementasikan setiap kelompok harus dikaitkan dengan fungsi objektif dari permainan ini, yaitu memenangkan permainan dengan memperoleh diamond sebanyak mungkin dan mencegah diamond diambil oleh bot lain.
4. Algoritma yang telah dibuat oleh mahasiwa, dijelaskan dan ditulis secara eksplisit di dalam laporan.
5. Program mengandung komentar yang jelas dan dilengkapi dengan kode sumber yang dibuat.
6. Program adalah program yang dibuat oleh mahasiswa.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Menurut (Vince, 2002, 247-248), algoritma *greedy* merupakan algoritma terbaik yang diketahui dalam memecahkan masalah kombinatorial. algoritma ini membuat pemecahan masalah yang optimal secara lokal. Algoritma ini bagus untuk mencari solusi dari masalah dengan ukuran data yang besar. Walaupun, pemecahan masalah yang dihasilkan oleh algoritma *greedy* tidak selalu menjadi solusi optimal secara global.

2.2 Cara Kerja Program

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

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

BAB III
APLIKASI STRATEGI GREEDY

3.1 Proses Mapping

Pada pembuatan bot permainan diamonds, algoritma greedy dibuat dengan pertama-tama melakukan mapping dari elemen pada permainan diamonds kepada elemen pada algoritma greedy

1. Himpunan kandidat (C) : Tiles yang bersebelahan dengan bot
2. Himpunan solusi, S : Tile yang bersebelahan dengan posisi bot dan memiliki evaluasi tertinggi
3. Fungsi solusi: Pemeriksaan apakah tile yang dipilih bersebelahan dengan posisi bot dan berada di board.
4. Fungsi seleksi (selection function): Memilih tiles yang meminimumkan fungsi objektif.
5. Fungsi kelayakan (feasible): Tiles yang bersebelahan merupakan valid. Pemeriksaan apakah ada faktor lain yang perlu dipertimbangkan selain jarak rata-rata diamonds (inventory, waktu, dll)
6. Fungsi objektif : f(tile)-> jarak rata-rata diamonds ke tile tersebut

3.2 Alternatif Solusi

1. DensityBot

Setiap kali bot Density ini perlu bergerak, ia pertama-tama akan menganalisis semua diamond yang ada di papan permainan. Diamond-diamond ini kemudian diurutkan berdasarkan sebuah metrik kepadatan nilai. Metrik ini dihitung dengan membagi poin yang ditawarkan oleh sebuah diamond dengan jarak Manhattan (jumlah langkah horizontal dan vertikal) dari posisi bot saat ini ke diamond tersebut. Jika jaraknya nol (bot berada di atas diamond), jarak tersebut dianggap satu untuk menghindari pembagian dengan nol. Pengurutan dilakukan secara menurun, sehingga diamond dengan rasio poin per jarak tertinggi—artinya yang paling menguntungkan untuk dikejar saat itu—menjadi target utama.

Setelah menentukan diamond target yang paling menarik, bot akan mengevaluasi serangkaian kondisi untuk memutuskan apakah akan melanjutkan pengejaran diamond atau beralih target untuk kembali ke base. Keputusan untuk kembali ke base akan diambil jika salah satu dari tiga kondisi terpenuhi: pertama, jika inventory bot sudah penuh; kedua, jika jumlah langkah yang dibutuhkan untuk mencapai base sama dengan sisa waktu permainan dalam detik dikurangi satu (memberikan sedikit margin waktu); atau ketiga, jika diamond target utama adalah diamond merah (2 poin) dan inventory bot hanya kurang satu slot untuk penuh. Kondisi terakhir ini mengindikasikan bahwa pengambilan diamond merah tersebut akan langsung memenuhi inventory, sehingga lebih bijak untuk segera mengamankan poin.

Jika salah satu kondisi untuk kembali ke base terpenuhi, bot akan menggunakan fungsi utilitas get_direction untuk menentukan langkah berikutnya menuju posisi base. Sebaliknya, jika tidak ada kondisi yang mengharuskan kembali ke base, bot akan menggunakan get_direction untuk bergerak menuju diamond target dengan kepadatan nilai tertinggi yang telah diidentifikasi sebelumnya. Dengan demikian, bot secara dinamis memilih antara memaksimalkan perolehan poin dari diamond terdekat yang bernilai tinggi atau mengamankan poin yang sudah ada di inventory ketika kondisi mendesak

2. WeightBot

Algoritma WeightBot ini dirancang untuk mengumpulkan diamond dengan fokus pada kedekatan, sambil memanfaatkan teleporter untuk efisiensi jalur dan memiliki kondisi kembali ke base yang strategis. Setiap kali bot mengambil keputusan, ia pertama-tama memperbarui informasi mengenai statusnya, posisinya, lokasi base, serta daftar diamond dan teleporter di papan. Jika tidak ada diamond yang tersedia, bot akan tetap diam. Untuk navigasi, bot menggunakan fungsi distance untuk menghitung jarak Manhattan standar dan fungsi distance with teleport yang lebih canggih untuk menentukan jarak terpendek dengan mempertimbangkan jalur langsung atau melalui pasangan teleporter.

bot akan selalu memilih diamond yang memiliki jarak terpendek dari posisinya saat ini, dengan jarak tersebut dihitung menggunakan distance with teleport, sehingga secara implisit memperhitungkan penggunaan teleporter jika itu menghasilkan jalur yang lebih pendek ke diamond. Setelah diamond target terdekat ditentukan, bot mengevaluasi apakah harus kembali ke base. Keputusan ini diambil jika inventory bot penuh, jika estimasi waktu perjalanan ke base (menggunakan distance_with_teleport) setidaknya sama dengan sisa waktu permainan, atau jika diamond target terdekat bernilai 2 poin dan akan langsung mengisi inventory bot yang hanya kurang satu slot. Berdasarkan kondisi ini, tujuan akhir bot ditetapkan ke base atau posisi diamond target terdekat. Selanjutnya, bot menentukan apakah penggunaan teleporter akan mempersingkat perjalanan ke tujuan akhir ini, dengan mekanisme untuk mencegah perulangan penggunaan teleporter yang sama. Target gerakan aktual kemudian ditetapkan, dan bot menghitung langkah berikutnya untuk bergerak satu unit menuju target tersebut, dengan memprioritaskan gerakan horizontal.

3. ProfitBot

Algoritma ProfitBot beroperasi dengan strategi utama yang berfokus pada perolehan diamond bernilai poin tertinggi, sambil mengintegrasikan penggunaan teleporter untuk efisiensi jalur dan serangkaian kondisi untuk kembali ke base. Setiap kali bergerak, bot pertama-tama mengidentifikasi statusnya, posisi, lokasi base, serta semua diamond dan teleporter di papan. Jika tidak ada diamond, bot tidak akan bergerak. Untuk navigasi, ia menggunakan dua fungsi jarak: satu menghitung jarak Manhattan standar, dan yang lebih canggih, distance_with_teleport,

menentukan jalur terpendek dengan mempertimbangkan rute langsung dan rute melalui pasangan teleporter yang tersedia. Inti dari strategi "Profit" ini adalah pemilihan target diamond, di mana bot secara eksklusif memilih diamond dengan nilai poin tertinggi (max(diamonds, key=lambda d: d.properties.points)), tanpa memperhitungkan jarak awal dalam keputusan ini.

ProfitBot mengevaluasi apakah harus kembali ke base. Keputusan ini dipicu jika inventory bot penuh, jika estimasi waktu perjalanan ke base (menggunakan distance_with_teleport) setidaknya sama dengan sisa waktu permainan, atau jika diamond target bernilai 2 poin dan akan langsung mengisi inventory bot yang hanya kurang satu slot. Berdasarkan kondisi ini, tujuan akhir bot ditetapkan ke base atau posisi diamond target. Selanjutnya, bot menentukan apakah penggunaan teleporter akan mempersingkat perjalanan ke tujuan akhir ini, dengan mekanisme untuk mencegah perulangan penggunaan teleporter yang sama secara berurutan. Target gerakan aktual kemudian ditetapkan, baik itu teleporter perantara atau tujuan akhir. Akhirnya, bot menghitung langkah konkret (delta_x, delta_y) untuk bergerak satu unit menuju target gerakan tersebut, dengan memprioritaskan gerakan horizontal.

3.3 Analisis Efisiensi Dan Efektivitas

Tiga strategi greedy Density, Weight, dan Profit menawarkan pendekatan berbeda untuk memandu bot dalam permainan mengumpulkan diamond. Strategi Density bekerja dengan menganalisis semua diamond di papan permainan dan mengurutkannya berdasarkan metrik kepadatan nilai, yang dihitung dengan membagi poin diamond dengan jarak Manhattan dari bot; diamond dengan rasio poin per jarak tertinggi menjadi target utama. Bot ini memutuskan kembali ke base jika inventory penuh, jika waktu yang dibutuhkan untuk mencapai base hampir sama dengan sisa waktu permainan, atau jika targetnya adalah diamond merah (2 point) dan inventory hanya kurang satu slot.

Strategi Weight memprioritaskan kembali ke markas jika bot telah mengumpulkan lima diamond. Jika belum, bot akan mencari diamond yang jaraknya terdekat dari bot. Bot tidak memikirkan point dari diamond dan hanya fokus mencari terdekat.

Strategi Profit mengidentifikasi semua objek game dan mengevaluasi diamond berdasarkan jumlah poinnya. Bot akan bergerak ke arah diamond dengan poin tertinggi. Kondisi kembali ke base meliputi inventory penuh, sisa waktu yang hanya cukup untuk perjalanan ke base, atau jika targetnya diamond merah sementara inventory sudah berisi empat diamond.

3.1 Strategi Greedy Yang Dipilih

Berdasarkan hasil 10 percobaan, di mana strategi Density menang 6 kali, Weight menang 5 kali, dan Profit menang 2 kali (seri dihitung sebagai kemenangan), strategi Density dipilih. Pemilihan ini didasarkan pada kinerja unggulnya dalam uji coba praktis, yang menunjukkan

bahwa pendekatan kepadatan nilai dikombinasikan dengan kondisi pulang yang seimbang terbukti paling sering berhasil. Strategi Density menawarkan keseimbangan yang baik antara agresivitas dalam mencari nilai per langkah tertinggi dan kehati-hatian melalui kondisi kembali ke base yang mempertimbangkan kapasitas inventory, sisa waktu kritis, dan situasi spesifik hampir penuh dengan diamond merah. Selain itu, Density memiliki efisiensi komputasi yang wajar dibandingkan Profit dan tidak memiliki kelemahan signifikan seperti kondisi pulang yang terlalu kaku pada Weight atau potensi konservatisme berlebih pada Profit. Kombinasi ini membuat Density menjadi pilihan yang paling efektif dalam skenario yang diuji.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

```
ALGORITMA DensityBot

INisialisasi:
  goal_position = TidakAda  // Posisi tujuan saat ini (diamond atau
base)
  last_teleporter_used = TidakAda // Menyimpan teleporter terakhir yang
digunakan untuk mencegah looping

FUNGSI next_move (board_bot, board):
  // --- Setup Awal ---
  bot_properties = board_bot.properties // Properti bot (inventory, sisa
waktu, posisi base)
  current_position = board_bot.position // Posisi bot saat ini
  base_position = bot_properties.base   // Posisi base bot
  all_diamonds_on_board = board.diamonds // Semua diamond yang ada di
papan

  // Jika tidak ada diamond di papan, jangan bergerak
  JIKA all_diamonds_on_board KOSONG MAKA
    KEMBALIKAN (0, 0) // Tidak ada pergerakan (delta_x = 0, delta_y = 0)
  AKHIR JIKA

  // Ambil objek teleporter dari papan (maksimal 2)
  all_game_objects = board.game_objects
  teleporters_list = []
  UNTUK setiap objek DALAM all_game_objects LAKUKAN
    JIKA objek.type == TeleportGameObject MAKA
      TAMBAHKAN objek KE teleporters_list
    AKHIR JIKA
  AKHIR UNTUK
  teleporter1 = teleporters_list[0] JIKA teleporters_list PUNYA
SETIDAKNYA 1 elemen, JIKA TIDAK TidakAda
  teleporter2 = teleporters_list[1] JIKA teleporters_list PUNYA
SETIDAKNYA 2 elemen, JIKA TIDAK TidakAda
```

```
  // --- Fungsi Bantuan untuk Perhitungan Jarak ---
  FUNGSI manhattan_distance (pos_a, pos_b):
    // Menghitung jarak Manhattan (jumlah langkah horizontal dan
vertikal)
    KEMBALIKAN absolut(pos_a.x - pos_b.x) + absolut(pos_a.y - pos_b.y)
  AKHIR FUNGSI

  FUNGSI distance_with_teleport (start_pos, end_pos):
    // Menghitung jarak terpendek, mempertimbangkan jalur langsung dan
via teleporter
    jarak_langsung = manhattan_distance(start_pos, end_pos)

    JIKA teleporter1 ADA DAN teleporter2 ADA MAKA
      // Jarak via teleporter pertama (masuk di tele1, keluar di tele2)
      jarak_via_tele1 = manhattan_distance(start_pos,
teleporter1.position) + manhattan_distance(teleporter2.position,
end_pos)
      // Jarak via teleporter kedua (masuk di tele2, keluar di tele1)
      jarak_via_tele2 = manhattan_distance(start_pos,
teleporter2.position) + manhattan_distance(teleporter1.position,
end_pos)
      KEMBALIKAN minimum(jarak_langsung, jarak_via_tele1,
jarak_via_tele2)
    LAINNYA
      // Jika tidak ada pasangan teleporter yang valid, hanya hitung
jarak langsung
      KEMBALIKAN jarak_langsung
    AKHIR JIKA
  AKHIR FUNGSI

  // --- Pemilihan Target Diamond Berdasarkan Kepadatan Nilai ---
  // Urutkan diamond berdasarkan kepadatan nilai (poin /
jarak_dengan_teleport)
  // Jarak minimum adalah 1 untuk menghindari pembagian dengan nol
  diamonds_sorted_by_density = URUTKAN all_diamonds_on_board berdasarkan
(
    lambda diamond: diamond.properties.points / maksimum(1,
distance_with_teleport(current_position, diamond.position))
```

```
  ) SECARA MENURUN

  // Target diamond adalah yang memiliki kepadatan nilai tertinggi
  target_diamond = diamonds_sorted_by_density[0]

  // --- Keputusan untuk Kembali ke Base ---
  remaining_time_seconds = INTEGER(bot_properties.milliseconds_left /
1000)
  distance_to_base = distance_with_teleport(current_position,
base_position)

  // Tentukan apakah bot harus kembali ke base
  should_return_to_base = (
    (bot_properties.diamonds == bot_properties.inventory_size) ATAU //
Inventory penuh
    (distance_to_base >= remaining_time_seconds) ATAU                //
Waktu hampir habis untuk kembali
    (target_diamond.properties.points == 2 DAN bot_properties.diamonds
== bot_properties.inventory_size - 1) // Mengambil diamond 2 poin akan
mengisi inv.
  )

  // Tetapkan tujuan akhir (destination) berdasarkan keputusan di atas
  JIKA should_return_to_base MAKA
    final_destination = base_position
  LAINNYA
    final_destination = target_diamond.position
  AKHIR JIKA

  // --- Keputusan untuk Menggunakan Teleporter untuk Pergerakan ---
  FUNGSI should_use_teleporter_for_move (from_pos, to_pos):
    // Memeriksa apakah menggunakan teleporter lebih pendek untuk
mencapai tujuan akhir
    JIKA BUKAN (teleporter1 ADA DAN teleporter2 ADA) MAKA
      KEMBALIKAN TidakAda // Tidak ada teleporter yang valid
    AKHIR JIKA

    jarak_langsung_ke_tujuan = manhattan_distance(from_pos, to_pos)
    jarak_via_tele1_ke_tujuan = manhattan_distance(from_pos,
```

```
teleporter1.position) + manhattan_distance(teleporter2.position, to_pos)
    jarak_via_tele2_ke_tujuan = manhattan_distance(from_pos,
teleporter2.position) + manhattan_distance(tele1.position, to_pos)

    // Pilih teleporter terdekat untuk dituju jika lebih efisien dan
bukan yang baru saja digunakan
    JIKA jarak_via_tele1_ke_tujuan < jarak_langsung_ke_tujuan DAN
teleporter1.position != last_teleporter_used MAKA
      KEMBALIKAN teleporter1.position
    LAIN JIKA jarak_via_tele2_ke_tujuan < jarak_langsung_ke_tujuan DAN
teleporter2.position != last_teleporter_used MAKA
      KEMBALIKAN teleporter2.position
    LAINNYA
      KEMBALIKAN TidakAda // Tidak ada jalur teleporter yang lebih baik
atau aman dari loop
    AKHIR JIKA
  AKHIR FUNGSI

  // Tentukan apakah akan melompat ke teleporter dalam perjalanan ke
tujuan akhir
  teleporter_jump_target =
should_use_teleporter_for_move(current_position, final_destination)

  // --- Penentuan Target Gerakan Aktual ---
  // move_target adalah posisi berikutnya yang dituju (bisa teleporter
atau tujuan akhir)
  JIKA teleporter_jump_target ADA MAKA
    // Jika diputuskan menggunakan teleporter, tuju teleporter tersebut
    last_teleporter_used = teleporter_jump_target // Ingat teleporter
ini untuk mencegah loop
    actual_move_target = teleporter_jump_target
  LAINNYA
    // Jika tidak menggunakan teleporter, langsung tuju tujuan akhir
    // Perbarui last_teleporter_used jika bot kebetulan berada di atas
teleporter
    // untuk mencegah penggunaan kembali instan yang tidak diinginkan
    JIKA teleporter1 ADA DAN current_position == teleporter1.position
MAKA
      last_teleporter_used = current_position
```

```
    LAIN JIKA teleporter2 ADA DAN current_position ==
teleporter2.position MAKA
      last_teleporter_used = current_position
    AKHIR JIKA
    actual_move_target = final_destination
  AKHIR JIKA

  // --- Penghitungan Delta Gerakan (Satu Langkah) ---
  // Hitung perbedaan koordinat ke target gerakan aktual
  delta_x_raw = actual_move_target.x - current_position.x
  delta_y_raw = actual_move_target.y - current_position.y

  // Batasi gerakan menjadi maksimal 1 unit per sumbu (langkah tunggal)
  delta_x = maksimum(minimum(delta_x_raw, 1), -1)
  delta_y = maksimum(minimum(delta_y_raw, 1), -1)

  // Prioritaskan gerakan horizontal: jika bergerak secara horizontal
(delta_x != 0),
  // maka jangan bergerak secara vertikal (delta_y = 0) di giliran yang
sama.
  JIKA delta_x != 0 MAKA
    delta_y = 0
  AKHIR JIKA

  KEMBALIKAN (delta_x, delta_y) // Kembalikan perubahan posisi untuk
langkah berikutnya
AKHIR FUNGSI
```

4.2 Struktur Data

Kelas/Objek/Fungsi	Atribut / Parameter	Tipe Data / Tipe Kembalian	Penjelasan
Kelas DensityBot			Kelas utama yang mendefinisikan perilaku bot.
	goal_position	Optional[Position]	Koordinat tujuan bot saat ini (diamond atau base). Bisa None.
	last_teleporter_used	Optional[Position]	Koordinat teleporter terakhir yang digunakan untuk mencegah looping. Bisa None.
	__init__(self)	None	Konstruktor kelas; menginisialisasi goal_position dan last_teleporter_used ke None.
	next_move(self, board_bot, board)	Tuple[int, int]	Metode inti yang berisi logika pengambilan keputusan bot untuk setiap giliran. Mengembalikan (delta_x, delta_y) untuk langkah berikutnya.
	board_bot (parameter)	GameObject	Objek yang merepresentasikan bot itu sendiri di dalam game.
	board (parameter)	Board	Objek yang merepresentasikan seluruh papan permainan.

Objek GameObject			Merepresentasikan entitas di papan permainan (bot, diamond, teleporter).
	position	Position	Koordinat (x, y) objek di papan.
	properties	Any	Properti spesifik objek. Untuk bot: base, diamonds, inventory_size, milliseconds_left. Untuk diamond: points.
	type	str	Jenis objek, misalnya, TeleportGameObject.
Objek Board			Merepresentasikan keseluruhan kondisi papan permainan.
	diamonds	List[GameObject]	Daftar semua objek diamond di papan.
	game_objects	List[GameObject]	Daftar semua objek di papan (bisa termasuk teleporter, dll.).
Objek Position			Merepresentasikan koordinat di papan permainan.
	x	int	Koordinat sumbu horizontal.
	y	int	Koordinat sumbu vertikal.
distance_with_teleport	start_pos, end_pos	int	Menghitung jarak terpendek antara dua posisi, mempertimbangkan penggunaan teleporter.

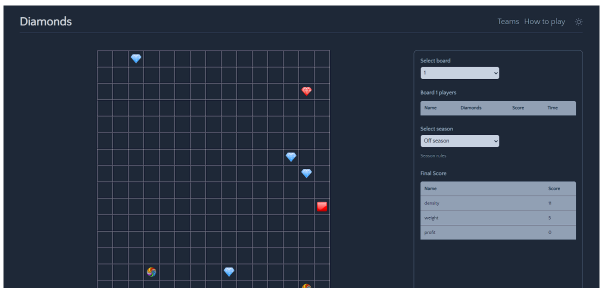
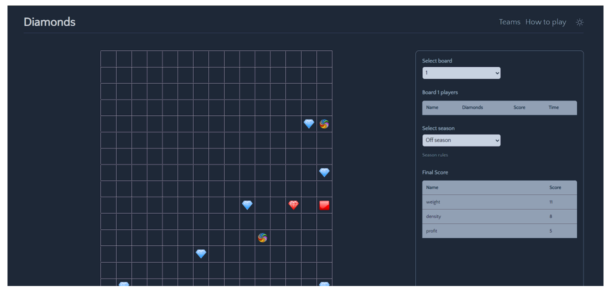
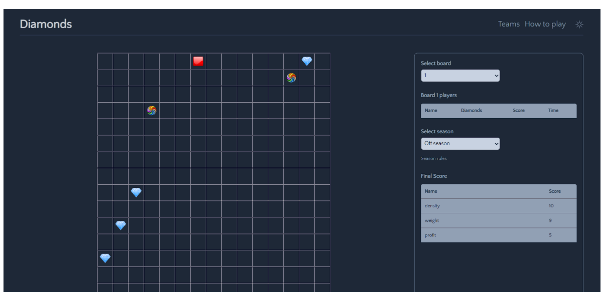
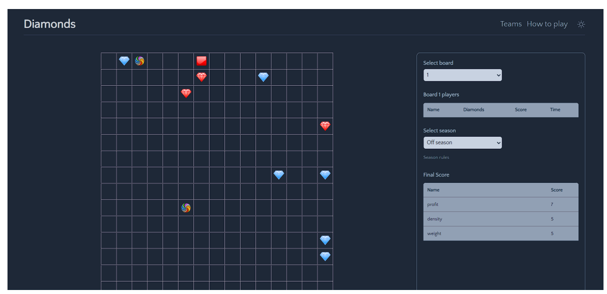
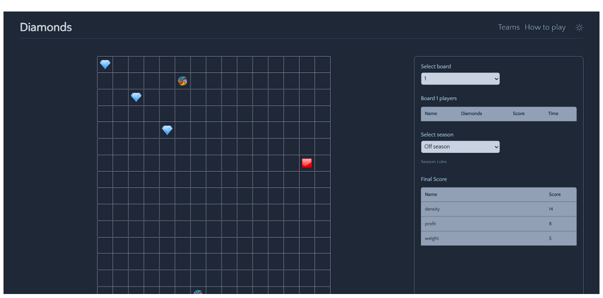
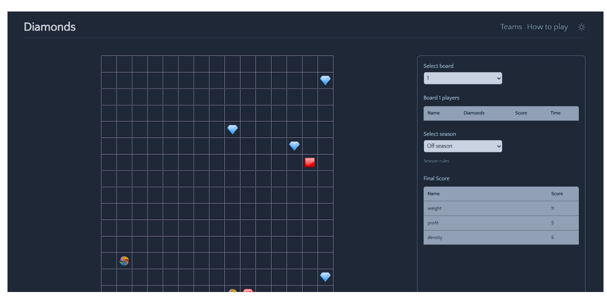
should_use_teleporter_for_move	from_pos, to_pos	Optional[Position]	Menentukan apakah menggunakan teleporter lebih pendek untuk mencapai tujuan dan mengembalikan posisi teleporter jika ya (memeriksa last_teleporter_used).

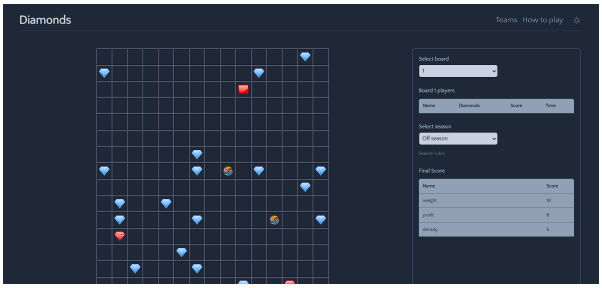
4.3. Analisis Desain Solusi Algoritma Greedy

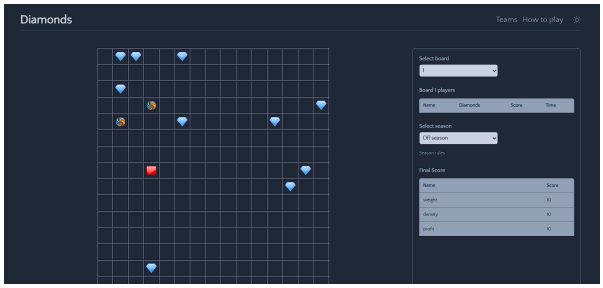
Algoritma DensityBot yang diimplementasikan dalam kode sumber bekerja dengan memprioritaskan diamond berdasarkan kepadatan nilai, yang dihitung sebagai rasio poin diamond dibagi dengan jarak terpendek ke diamond tersebut, di mana jarak ini secara cerdas mempertimbangkan penggunaan teleporter melalui fungsi distance with teleport. Bot akan memilih diamond dengan kepadatan nilai tertinggi sebagai target utamanya. Keputusan untuk kembali ke base dipicu jika inventory penuh, jika estimasi waktu perjalanan ke base (juga menggunakan distance_with_teleport) sama atau melebihi sisa waktu permainan, atau jika target diamond saat ini bernilai 2 poin sementara inventory hanya kurang satu slot untuk penuh. Untuk navigasi, bot juga menentukan apakah penggunaan teleporter adalah jalur yang lebih efisien untuk mencapai tujuannya (baik itu diamond atau base), dan menyertakan mekanisme last teleporter used untuk mencegah perulangan sederhana antar teleporter. Gerakan aktual dilakukan selangkah demi selangkah, dengan prioritas pada sumbu horizontal.

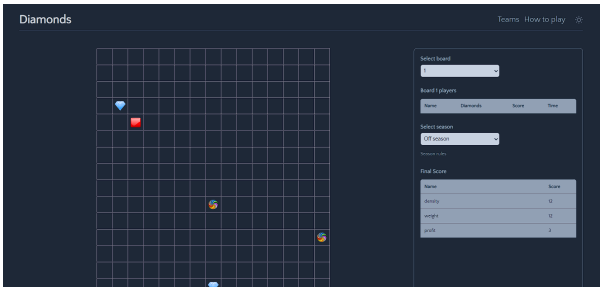
Strategi greedy ini cenderung berhasil mendapatkan hasil yang mendekati optimal pada peta dengan diamond yang terdistribusi baik dan teleporter yang ditempatkan secara strategis, terutama ketika gangguan dari lawan minim, karena memungkinkan bot memaksimalkan perolehan poin per unit waktu tempuh. Manajemen waktu kembali ke base juga cukup baik dalam mengamankan poin. Namun, kelemahan utamanya, seperti yang telah teridentifikasi, adalah ketidakmampuannya untuk memperhitungkan atau menghindari manuver lawan, sehingga rentan terhadap teckle atau blokade. Selain itu, fokus pada optimal lokal (diamond terpadat saat ini) terkadang bisa menjebak bot menjauhi area yang lebih kaya secara global atau mengabaikan diamond dengan kepadatan sedikit lebih rendah yang strategis. Meskipun demikian, dalam 10 pengujian yang dilakukan, strategi Density berhasil meraih kemenangan terbanyak, walau dengan selisih tipis dari strategi Weight, menunjukkan bahwa efisiensi pengumpulan poinnya

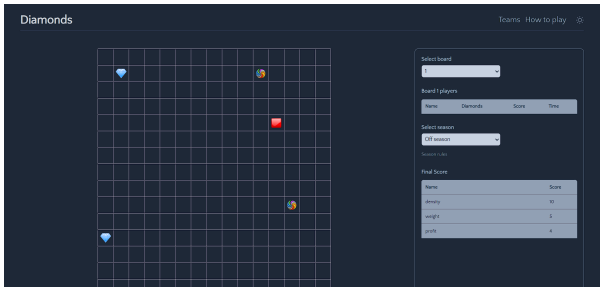
seringkali mampu mengimbangi kerugian akibat belum adanya penghindaran lawan. Pengujian lebih lanjut dalam skenario unik, seperti peta dengan lawan yang sangat padat dan agresif, konfigurasi teleporter yang rumit, atau kondisi akhir permainan yang kritis, akan sangat berguna untuk mengeksplorasi batas-batas efektivitas strategi ini dan mengidentifikasi potensi perbaikan, terutama dalam hal kesadaran terhadap lingkungan dan interaksi dengan bot lain.

Screenshot Pengujian Density vs Weight vs Profit	
	
	
	


Diamonds
Teams
How to play
Select board
Board 1 players
Name
Diamonds
Score
Time
Select season
Off season
Final Score
Name
Score


Diamonds
Teams
How to play
Select board
Board 1 players
Name
Diamonds
Score
Time
Select season
Off season
Final Score
Name
Score


Diamonds
Teams
How to play
Select board
Board 1 players
Name
Diamonds
Score
Time
Select season
Off season
Final Score
Name
Score


Diamonds
Teams
How to play
Select board
Board 1 players
Name
Diamonds
Score
Time
Select season
Off season
Final Score
Name
Score

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Strategi greedy yang diimplementasikan cukup efektif dalam memaksimalkan pengumpulan poin dengan memprioritaskan diamond berdasarkan kepadatan nilai yang mempertimbangkan poin dan jarak, termasuk penggunaan teleporter secara cerdas. Bot ini juga memiliki mekanisme yang baik untuk kembali ke base, terutama dengan memperhitungkan sisa waktu dan kondisi inventory yang hampir penuh saat menargetkan diamond bernilai tinggi. Keberhasilannya meraih kemenangan terbanyak dalam 10 pengujian, meskipun tipis dan masih rentan terhadap teckel dari lawan, menunjukkan bahwa efisiensi pengumpulan poinnya seringkali dapat mengimbangi ketiadaan mekanisme penghindaran lawan.

5.2 Saran

Untuk pengembangan lebih lanjut, disarankan agar DensityBot dilengkapi dengan kemampuan untuk mendeteksi dan merespons keberadaan serta pergerakan bot lawan. Implementasi strategi penghindaran sederhana, seperti tidak menargetkan diamond yang jelas-jelas akan diambil lebih dulu oleh lawan atau menjaga jarak aman, dapat secara signifikan mengurangi kerentanan terhadap teckel dan meningkatkan tingkat keberhasilan secara keseluruhan. Selain itu, pengujian yang lebih ekstensif dengan variasi peta dan jumlah lawan yang lebih banyak akan memberikan data yang lebih komprehensif untuk menyempurnakan logika pengambilan keputusan, terutama dalam situasi yang lebih kompleks atau kompetitif, serta untuk mengevaluasi lebih lanjut robusticity mekanisme pencegahan looping teleporter. Pertimbangan untuk menambahkan bobot risiko atau potensi gangguan dari lawan dalam perhitungan kepadatan nilai juga bisa menjadi langkah maju untuk pengambilan keputusan yang lebih canggih.

LAMPIRAN

Link Github : https://github.com/Gahehe52/Tubes-Stigma

Link Video :
https://drive.google.com/drive/folders/1EmL5S4xnGKu0Bp7_iWdwkuCkykuarwUI?usp=sharing

DAFTAR PUSTAKA

Vince, A. (2002, September 15). A framework for the greedy algorithm. *Discrete Applied Mathematics*, *121*(1-3), 247-260. ScienceDirect. https://doi.org/10.1016/S0166-218X(01)00362-6

Yizhun, W. (2023, November 30). Review on greedy algorithm. *Theoretical and Natural Science*, *14*, 233-239. EWA Publishing. https://doi.org/10.54254/2753-8818/14/20241041