

IMPLEMENTASI ALGORITMA X DALAM PEMECAHAN BOT PERMAINAN DIAMOND

Tugas Besar

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



Oleh: Kelompok 13 (ANT Squad)

M. Arif Ardani 123140186

Nadya Shafwah Yusuf 123140167

Fadina Mustika Ratnaningsih 123140157

Dosen Pengampu: Imam Eko Wicaksono, S.Si., M.Si.

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

DAFTAR ISI

BAB I.....	3
DESKRIPSI TUGAS.....	3
BAB II.....	4
LANDASAN TEORI.....	4
2.1 Dasar Teori.....	4
2.2 Cara Kerja Program.....	4
2.2.1 Cara Implementasi Program.....	5
2.2.2 Menjalankan Bot Program.....	5
2.2.3 Strategi Bot dan Mekanisme Interaksi.....	6
BAB III.....	7
APLIKASI STRATEGI GREEDY.....	7
3.1 Proses Mapping.....	7
3.2 Eksplorasi Alternatif Solusi Greedy.....	7
3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy.....	10
3.4 Strategi Greedy yang Dipilih.....	11
BAB IV.....	14
IMPLEMENTASI DAN PENGUJIAN.....	14
4.1 Implementasi Algoritma Greedy.....	14
4.1.1 Pseudocode.....	14
4.1.2 Penjelasan Alur Program.....	26
4.2 Struktur Data yang Digunakan.....	27
4.3 Pengujian Program.....	27
4.3.1 Skenario Pengujian.....	27
4.4 Hasil Pengujian dan Analisis.....	28
BAB V.....	30
KESIMPULAN DAN SARAN.....	30
5.1 Kesimpulan.....	30
5.2 Saran.....	30
LAMPIRAN.....	32
DAFTAR PUSTAKA.....	33

BAB I

DESKRIPSI TUGAS

Permainan Diamonds adalah sebuah *programming challenge* di mana harus mempertandingkan bot yang kami buat dengan bot yang dibuat oleh pemain lainnya. Setiap pemain akan memiliki sebuah bot yang tujuannya adalah mengumpulkan sebanyak mungkin diamond. Setiap pemain harus menerapkan strategi tertentu pada setiap bot mereka untuk memenangkan pertandingannya. Permainan ini terdiri dari *Game Engine*, yang berisikan kode *backend* dan *frontend* permainan dan *Bot Starter Pack*, yang berisi program untuk memanggil API, program *bot logic*, dan program utama/utilitas lainnya. Program *bot logic* inilah yang akan diimplementasikan dengan algoritma Greedy.

Untuk memenangkan pertandingan, Anda harus mengumpulkan diamond ini sebanyak mungkin dan melewatkinya. Diamond merah dan biru masing-masing bernilai dua poin, sedangkan yang biru bernilai satu poin. Setiap kali diamond *regeneration*, rasio antara diamond merah dan biru akan berubah. Anda harus menggerakkan bot dalam game ini untuk mendapatkan banyak diamond. Semua bot memiliki base yang dapat digunakan untuk menyimpan diamond yang dibawa. Apabila diamond disimpan ke base, score bot akan meningkat sesuai dengan nilai diamond yang dibawa, dan inventory bot akan menjadi kosong. Inventory ini memiliki kapasitas tertinggi, jadi sewaktu-waktu bisa penuh. Jika inventory ini tidak penuh, bot dapat menyimpan isi ke base agar inventory bisa kosong kembali.

BAB II

LANDASAN TEORI

2.1 Dasar Teori

Algoritma Greedy adalah algoritma yang menyelesaikan masalah optimasi langkah demi langkah untuk menemukan solusi terbaik. Namun, setiap solusi jawaban dari algoritma ini tidak selalu merupakan solusi optimum global, tetapi algoritma ini dapat diandalkan untuk mengoptimasi waktu [1]. Pada algoritma greedy, persoalan optimasi terdiri dari elemen-elemen Himpunan kandidat (C), yang mengandung elemen-elemen pembentuk solusi, dan kemudian Himpunan solusi (S), yang mengandung himpunan dari kandidat-kandidat yang dipilih untuk menyelesaikan persoalan. Fungsi seleksi terjadi pada setiap langkah untuk memilih kandidat terbaik untuk mencapai hasil terbaik. Fungsi kelayakan (feasible) menentukan apakah kandidat yang dipilih dapat memberikan solusi yang layak. Fungsi obyektif menentukan nilai solusi dengan memaksimumkan atau memnimumkan nilainya [2].

Dengan kata lain, algoritma greedy melibatkan pencarian himpunan bagian, S , dari himpunan kandidat, C . Dalam algoritma ini, S harus memenuhi beberapa kriteria, seperti menyatakan solusi, dan dioptimisasi oleh fungsi objektif. Algoritma greedy tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada, dan juga memiliki beberapa fungsi seleksi yang berbeda, sehingga harus memilih fungsi yang tepat jika ingin algoritma menghasilkan solusi optimum. Akibatnya, algoritma global optimum belum tentu merupakan solusi optimum (terbaik), tetapi suboptimum atau pseudooptimum [3].

2.2 Cara Kerja Program

Untuk menghindari bot lawan dan menggunakan fitur khusus seperti teleporter dan red button, bot-bot akan bergerak di board untuk mengumpulkan diamond dan mengembalikannya ke home base. Dengan melakukan ini, mereka dapat menambah skor. Permainan berlangsung dalam waktu terbatas, dan setelah waktu habis, skor akhir ditentukan. Berikut penjabaran tentang program, antara lain:

2.2.1 Cara Implementasi Program

Cara kerja program ini adalah dengan menggunakan *Starter Bot Pack* yang tersedia. Program bot akan berinteraksi dengan Game Engine melalui *Application Programming Interface (API)* yang tersedia. Bot selalu memanggil API untuk mendapatkan data terbaru tentang kondisi board permainan. Informasi ini mencakup posisi diamond yaitu biru dan merah, red button, keberadaan dan lokasi teleporter, lokasi bot lawan, dan home base bot.

Bot menggunakan algoritma Greedy untuk menentukan langkah atau tindakan terbaik untuk setiap putaran berdasarkan data yang diterima. Konsep Greedy di sini berarti bot mengambil keputusan lokal yang paling ideal untuk mencapai skor total maksimum. Misalnya, bot menghitung poin diamond potensial yang dapat diperoleh dari setiap langkah. Selain itu, karena bot memiliki kapasitas penyimpanan yang terbatas, bot akan memprioritaskan kembali ke home base untuk menyimpan diamond dan mengosongkan penyimpanan ketika penyimpanan hampir penuh. Kemungkinan berinteraksi dengan bot lawan juga dapat memengaruhi logika Greedy. Misalnya, kemungkinan berinteraksi dengan bot lawan berarti bot dapat kehilangan diamond dari home base bot jika ditabrak oleh lawan, atau bahkan melakukan tabrakan jika itu menguntungkan.

2.2.2 Menjalankan Bot Program

Terdapat dua komponen utama yang harus disiapkan sebelum program bot permainan Diamonds dapat dijalankan yaitu Game Engine dan program bot yang telah diimplementasikan. Game Engine yang terdiri dari kode frontend untuk visualisasi permainan dan kode backend yang mengelola logika permainan secara keseluruhan. Program Bot adalah program Python yang mengimplementasikan algoritma Greedy yang dibuat oleh kelompok. Program bot dapat dijalankan secara terpisah setelah Game Engine berjalan dan dapat terhubung ke Game Engine melalui API. Selama permainan, bot akan secara berkala mengirimkan aksi ke Game Engine, seperti pergerakan ke arah tertentu, dan menerima data board terbaru untuk membuat keputusan selanjutnya. Saat permainan berakhir, proses ini berhenti.

2.2.3 Strategi Bot dan Mekanisme Interaksi

Bot akan membuat keputusan berdasarkan rangkaian prioritas dan perhitungan keuntungan sesaat, yang merupakan dasar dari pendekatan greedy. Nantinya, bot akan memprioritaskan mengambil diamond merah karena diamond merah memiliki nilai lebih tinggi, yaitu 2 poin, dibandingkan diamond biru yang memiliki 1 poin. Menghitung jarak dan jalur terpendek menuju diamond dengan nilai tertinggi akan menjadi bagian penting dari penentuan strategi. Bot akan terus memantau kapasitas inventory-nya. Ketika kapasitasnya mendekati batasnya, bot akan mengubah prioritasnya dari mencari diamond menjadi bergerak ke home base untuk menyimpan diamond. Proses penyimpanan ini akan menambahkan skor bot dan membuat inventory dikosongkan kembali.

Kehadiran red button akan dipertimbangkan. Jika lokasi diamond saat ini tidak menguntungkan atau jika regeneration dari diamond baru mungkin menawarkan peluang yang lebih baik, bot mungkin akan memprioritaskan melangkah ke red button. Namun, berarti posisi red button akan berubah secara acak setelah dilangkahi, tergantung pada strategi Greedy yang dipilih. Selain itu, bot akan mengidentifikasi posisi teleporter dan mempertimbangkannya sebagai rute pintas untuk mencapai diamond atau base dengan lebih cepat. Lalu, mereka akan mempertimbangkan efisiensi waktu perjalanan melalui teleporter saat membuat keputusan Greedy.

Interaksi dengan bot lawan adalah salah satu kesulitan. Logika untuk melakukan dan menghindari tackle dapat menjadi bagian dari algoritma Greedy. Jika bot lawan berada di jalur yang sama atau mungkin melakukan tackle, bot mungkin memilih jalur lain untuk menghindari kehilangan diamond yang sudah dikumpulkan. Jika bot lawan berada dalam posisi yang menguntungkan untuk ditabrak, seperti memiliki banyak diamond di inventory, bot mungkin memprioritaskan untuk melakukan tackle agar dapat mengambil diamond lawan dan menambah skornya sendiri. Oleh karena itu, program secara keseluruhan menggunakan logika greedy untuk menentukan tindakan terbaik, dengan tujuan akhir untuk memaksimalkan perolehan diamond dan skor dalam waktu yang singkat.

BAB III

APLIKASI STRATEGI GREEDY

3.1 Proses *Mapping*

Proses mapping adalah langkah awal, di mana masalah pengumpulan diamond dan pergerakan bot dalam permainan diinterpretasikan sesuai dengan prinsip algoritma greedy. Dalam permainan Diamonds, tujuan utama bot adalah memperoleh skor yang paling banyak dengan mengumpulkan sebanyak mungkin diamond. Setiap diamond memiliki nilai poin tertentu yaitu diamond merah memiliki dua poin, diamond biru memiliki satu poin, dan inventory bot memiliki kapasitas terbatas. Setiap langkah atau giliran bot memiliki beberapa pilihan aksi dengan bergerak ke atas, bawah, kiri, atau kanan. Karena bot hanya mempertimbangkan lingkungan sekitarnya saat ini, pilihan ini bersifat lokal. Perhitungan nilai atau prioritas dari setiap pilihan pergerakan akan menentukan kriteria pemilihan Greedy. Misalnya, jika inventory hampir penuh, diamond merah dan biru akan memiliki pergerakan yang lebih berat menuju base. Selain itu, jarak akan memengaruhi, karena tujuan yang lebih dekat dengan asumsi nilai keuntungan yang sama akan lebih diutamakan.

3.2 Eksplorasi Alternatif Solusi Greedy

1. Greedy by Return (Waktu Kritis):

Kondisi Pemicu: `remaining_time_seconds < 10 AND props.diamonds > 0 AND props.base`. Ini adalah strategi dengan prioritas tertinggi.

Jika waktu permainan hampir habis misal kurang dari 10 detik dan bot memiliki diamond di inventarisnya, bot akan segera memprioritaskan untuk kembali ke base untuk menyetor diamond dan mengamankan poin. Tujuannya adalah untuk mencegah kehilangan diamond yang sudah dikumpulkan karena waktu habis. Ini sangat efisien karena langsung mencari jalur tercepat ke base. Bot menghitung jalur terpendek ke base dan segera melakukan gerakan pertama pada jalur tersebut. Jika tidak ada jalur, ia akan mencoba gerakan acak yang valid.

2. Greedy by Escape (Menghindari Bot Lain):

Kondisi Pemicu: props.diamonds > self.EVADE_DIAMOND_THRESHOLD (saat ini 2) AND other_bots terdeteksi dalam jarak tertentu (saat ini 3 langkah).

Jika bot memiliki jumlah diamond di atas ambang batas tertentu (saat ini 2 diamond), dan mendeteksi bot lain di sekitarnya, bot akan mencoba menghindari bot lawan. Tujuannya adalah untuk mengurangi risiko kehilangan diamond yang sudah dikumpulkan akibat tackle dari bot lawan. Ini adalah keputusan defensif otomatis. Bot mencari sel tetangga yang valid yang memiliki jarak Manhattan terjauh dari bot lawan terdekat, dan bergerak menuju sel tersebut.

3. Greedy by Tackle:

Kondisi Pemicu: props.can_tackle adalah True, props.diamonds < 5 (tidak sedang penuh), DAN ada bot lawan dalam jangkauan 1 langkah (distance_map[other_bot_coord] == 1) yang memiliki diamond di atas ambang batas self.TACKLE_DIAMOND_THRESHOLD (saat ini 3 diamond).

Jika bot memiliki kemampuan untuk tackle dan tidak sedang membawa diamond dalam jumlah penuh, ia akan mencari bot lawan yang berada dalam jangkauan satu langkah dan memiliki diamond dalam jumlah signifikan. Prioritasnya adalah untuk mendapatkan diamond tambahan dengan cepat dari lawan. Efektivitas strategi ini sangat bergantung pada posisi relatif musuh. Bot akan langsung bergerak ke posisi bot lawan yang menjadi target tackle.

4. Greedy by Return (Inventori Penuh):

Kondisi Pemicu: props.diamonds >= 5 AND props.base.

Jika inventaris diamond bot sudah penuh (5 diamond atau lebih), bot akan memprioritaskan untuk kembali ke markas (base) untuk menyetorkan diamond. Ini adalah kondisi standar untuk mengamankan poin. Bot mencari jalur terpendek ke base.

5. Greedy by Return (Inventori Hampir Penuh):

Kondisi Pemicu: * props.diamonds == 4.

Ketika bot memiliki 4 diamond, ia akan mencoba mencari diamond biru (biasanya bernilai 1) sebagai diamond terakhir. Jika tidak ada diamond biru yang terjangkau, atau

jika base lebih dekat, bot akan memprioritaskan kembali ke base untuk segera menyetor diamond yang sudah ada. Ini mengurangi risiko kehilangan diamond saat hampir mencapai target penuh. Bot mencari diamond biru terdekat dengan prioritas jarak, lalu kembali ke base jika tidak ada diamond biru yang optimal.

6. Greedy by Diamond (dari Base):

Kondisi Pemicu: Tidak ada kondisi di atas yang terpenuhi AND props.base ada.
Bot akan mencoba menemukan diamond yang paling dekat dengan markasnya sendiri. Strategi ini membantu bot untuk membersihkan area di sekitar base dan tetap relatif dekat dengan titik setoran. Menghitung jarak diamond dari base dan memilih yang terdekat.

7. Greedy by Density (Utilitas Terbaik):

Kondisi Pemicu: Tidak ada kondisi di atas yang terpenuhi.
Bot akan mencari diamond dengan "utilitas terbaik" (nilai diamond dibagi jarak). Ini mengutamakan diamond merah (poin lebih tinggi) terlebih dahulu, kemudian diamond biru. Tujuannya adalah untuk memaksimalkan perolehan poin per langkah. Memilih diamond merah dengan rasio nilai/jarak terbaik, kemudian diamond biru dengan rasio nilai/jarak terbaik.

8. Greedy by Red Button:

Kondisi Pemicu: board.diamonds kosong.
Jika tidak ada diamond yang tersedia di papan permainan, bot akan memprioritaskan untuk bergerak menuju ke red button di mana artifact yang diidentifikasi sebagai generator diamond untuk memicu munculnya diamond baru. Bot mencari artifact dengan tipe 'artifact' dan bergerak menuju yang terdekat.

9. Greedy by Return (Fallback ke Base):

Kondisi Pemicu: Tidak ada target diamond atau red button yang ditemukan DAN props.base ada.

Sebagai langkah fallback, jika tidak ada diamond atau red button yang bisa dituju, bot akan kembali ke base. Hal ini memastikan bot tidak berkeliaran tanpa tujuan. Bot hanya mencari jalur terpendek ke base.

10. Greedy by Exploration (Unvisited Cells):

Kondisi Pemicu: Tidak ada target lain yang ditemukan pada diamond, base, red button.

Ini adalah strategi fallback terakhir. Jika tidak ada tujuan yang jelas yaitu tidak ada diamond, tidak ada base yang relevan, tidak ada red button, dan bot akan mencoba bergerak menuju sel terdekat yang belum pernah ia kunjungi. Tujuannya adalah untuk menjelajahi peta dan menemukan sumber daya baru. Bot mencari sel terdekat yang tidak ada di self.visited_cells.

11. Gerakan Acak (Fallback Akhir):

Kondisi Pemicu: Jika semua strategi di atas gagal menemukan target yang bisa dijangkau. Jika bot tidak dapat menentukan gerakan yang optimal berdasarkan prioritas di atas misalnya, semua jalur terblokir, atau tidak ada target yang valid, bot akan melakukan gerakan acak yang valid dengan satu langkah ke arah mana pun yang memungkinkan. Ini mencegah bot stuck. Bot mencoba gerakan ke arah (1,0), (-1,0), (0,1), (0,-1) secara berurutan dan memilih yang valid pertama.

3.3 Analisis Efisiensi dan Efektivitas Solusi Greedy

1. Greedy by Return (Waktu Kritis)

Saat waktu kurang dari 10 detik dan bot memiliki diamond di inventornya, maka bot akan mengutamakan kembali ke base untuk meminimalisir kerugian score karena diamond masih ada di inventori dan belum tersimpan di base.

2. Greedy by Escape (Menghindari Bot Lain)

Ketika bot memiliki lebih dari 2 diamond dalam inventori dan mendeteksi kehadiran bot lain, maka bot akan menghindari bot tersebut. Di saat yang bersamaan, bot akan mencoba kembali ke base untuk menyimpan diamond yang ada di inventori, sehingga risiko mengalami kerugian score karena di tackle bot lain akan diminimalisir.

3. Greedy by Tackle

Algoritma greedy by tackle digunakan untuk mencegah bot lain membawa diamond di inventori mereka ke dalam base. Sehingga, bot bisa mencegah bot lain memiliki score yang lebih tinggi.

4. Greedy by Return (Inventori Penuh)

Ketika bot telah mencapai inventori maksimum (5 diamond), maka bot akan langsung kembali ke base. Setelah itu baru melanjutkan pencarian diamond.

5. Greedy by Return (Inventori Hampir Penuh)

Ketika inventori bot sudah memiliki 4 diamond, maka bot akan mengutamakan untuk mencari blue diamond di sekitar. Jika tidak ada blue diamond di sekitar, maka bot akan memilih untuk kembali ke base untuk menyimpan dulu diamond yang ada di inventori. Algoritma ini mencegah bot stuck ketika sudah ada 4 diamond di inventori dan bertemu dengan red diamond.

6. Greedy by Red Button

Red button adalah tombol yang jika dilewati bot akan melakukan reset posisi diamond dan jumlahnya. Jika diamond tersisa sedikit dan lokasi lebih jauh dari base ketimbang dengan lokasi red button ke base, maka bot akan menghampiri red button, sehingga bisa melakukan reset posisi diamond dan memangkas waktu yang ditempuh jika bot menghabiskan diamond yang ada terlebih dahulu baru menunggu diamond lain bermunculan.

7. Greedy by Density

Greedy by density merupakan algoritma "pengumpul" utama yang efisien. Algoritma ini mengutamakan diamond dengan nilai atau jarak terbaik mulai dari merah, lalu biru dengan memastikan bot selalu berusaha mendapatkan diamond yang paling menguntungkan. Strategi diamond terdekat dari base juga mendukung efisiensi karena bot tidak terlalu jauh dari titik setoran.

8. Greedy by Exploration (Unvisited Cells)

Algoritma ini digunakan untuk mencegah bot *stuck* di satu tempat. Dengan algoritma ini, bot akan menjelajahi arah yang belum dieksplorasi, sehingga memperbesar peluang bot untuk menemukan lebih banyak diamond.

3.4 Strategi Greedy yang Dipilih

Berdasarkan analisis, dipilih kombinasi strategi dengan sistem prioritas bertingkat yang memungkinkan bot untuk beradaptasi dengan kondisi permainan yang berubah, memaksimalkan

perolehan poin dalam waktu terbatas, dan meminimalkan risiko kehilangan diamond. Berikut adalah kombinasi algoritma greedy yang dipilih :

1. Greedy by Return (Waktu Kritis)

Algoritma ini adalah cara efisien dalam mengamankan poin di akhir permainan.

Tidak peduli seberapa banyak diamond yang bisa didapat lagi, jika waktu kurang dari 10 detik dan bot memiliki diamond, satu-satunya tujuan efisien adalah menyetorkannya. Prioritas absolut ini mencegah kerugian poin yang tidak perlu.

2. Greedy by Escape (Menghindari Bot Lain)

Algoritma ini memiliki prioritas untuk mempertahankan aset (diamond). Jika bot telah mengumpulkan sejumlah diamond dan ada ancaman dari bot lawan, maka bot akan menghindari konfrontasi yang bisa menyebabkan kehilangan diamond.

3. Greedy by Tackle

Algoritma ini membantu untuk mencegah bot lain memiliki score yang lebih tinggi, mengingat tackle dapat menghilangkan diamond yang ada di inventori bot.

4. Greedy by Return (Inventori Penuh)

Sama seperti waktu kritis, ini adalah efisiensi dalam mengamankan poin. Begitu bot mencapai kapasitas diamond, prioritas tertinggi adalah menyetorkannya ke base sesegera mungkin untuk mengubahnya menjadi poin dan mengosongkan inventori untuk diamond berikutnya.

5. Greedy by Return

Ketika di dalam inventori terdapat 4 diamond, bot akan mencari blue diamond terdekat adalah cara efisien untuk cepat mencapai 5 diamond. Jika tidak ada blue diamond yang mudah dijangkau, kembali ke base adalah pilihan prioritas berikutnya untuk tidak membuang waktu.

6. Greedy by Red Button

Jika tidak ada diamond terdekat atau jumlah diamond sudah berkurang dna red button ada di jarak yang lebih dekat dengan bot, bot akan mengunjungi red button sehingga terjadi reset posisi diamond dan memunculkan lebih banyak diamond di tempat random.

7. Greedy by Density

Ini adalah algoritma "pengumpul" utama yang efisien. Mengutamakan diamond

dengan nilai/jarak terbaik (mulai dari merah, lalu biru) memastikan bot selalu berusaha mendapatkan diamond yang paling menguntungkan. Strategi "diamond terdekat dari base" juga mendukung efisiensi karena bot tidak terlalu jauh dari titik setoran.

8. Greedy by Exploration (Unvisited Cells)

Algoritma ini digunakan untuk mencegah bot *stuck* di area yang sudah dieksplorasi. Ketika di sekitar bot tidak ada diamond yang terdeteksi, maka bot bisa bergerak ke arah yang belum dieksplorasi oleh bot, sehingga memperbesar kemungkinan bot untuk menjangkau lebih banyak diamond.

BAB IV

IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Algoritma Greedy

4.1.1 Pseudocode

```
CLASS BaseLogic (Abstract)
    FUNCTION next_move(board_bot: GameObject, board: Board) ->
        Tuple(Integer, Integer)
            RAISE NotImplemented
    END FUNCTION
END CLASS

CLASS tripleN INHERITS BaseLogic
    // Atribut Inisialisasi
    PRIVATE visited_cells: Set of (Integer, Integer)
    PRIVATE current_target_position: Optional Position
    PRIVATE current_target_diamond_id: Optional String
    PRIVATE TACKLE_DIAMOND_THRESHOLD: Integer = 3
    PRIVATE EVADE_DIAMOND_THRESHOLD: Integer = 2
    PRIVATE EVADE_DETECTION_RANGE: Integer = 3

    FUNCTION CONSTRUCTOR()
        SET visited_cells TO empty Set
        SET current_target_position TO NULL
        SET current_target_diamond_id TO NULL
    END FUNCTION

    // Fungsi untuk menghitung jarak terpendek ke semua sel yang
    dapat dijangkau (BFS)
    FUNCTION _calculate_all_distances(start: Position, board: Board)
        -> Tuple(Map, Map)
        SET queue TO new Deque containing (start, 0)
        SET distance_map TO Map with (start.x, start.y) -> 0
        SET parent_map TO empty Map
        SET visited_bfs TO new Set containing (start.x, start.y)
```

```

SET possible_moves TO [(0, 1), (0, -1), (1, 0), (-1, 0)]

WHILE queue IS NOT EMPTY
    SET current_pos, dist TO queue.POP_LEFT()
    FOR EACH dx, dy IN possible_moves
        SET next_x, next_y TO current_pos.x + dx,
        current_pos.y + dy
        SET next_pos_obj TO new Position(y=next_y, x=next_x)
        SET next_pos_coord TO (next_x, next_y)

        IF board.is_valid_move(current_pos, dx, dy) AND
        next_pos_coord IS NOT IN visited_bfs THEN
            ADD next_pos_coord TO visited_bfs
            SET distance_map[next_pos_coord] TO dist + 1
            SET parent_map[next_pos_coord] TO
            (current_pos.x, current_pos.y)
            ADD (next_pos_obj, dist + 1) TO queue
        END IF
    END FOR
END WHILE

RETURN distance_map, parent_map
END FUNCTION

// Fungsi untuk merekonstruksi jalur gerakan dari parent_map
FUNCTION _reconstruct_path_moves(start: Position, target:
Position, parent_map: Map) -> Optional List of Tuple(Integer,
Integer)
    SET path_positions TO empty List
    SET target_coord TO (target.x, target.y)
    SET current_coord TO (target.x, target.y)
    SET start_coord TO (start.x, start.y)

    IF target_coord IS NOT IN parent_map AND current_coord IS
    NOT EQUAL TO start_coord THEN
        RETURN NULL
    END IF

```

```

        WHILE current_coord IS NOT EQUAL TO start_coord
            ADD new Position(x=current_coord[0], y=current_coord[1])
        TO path_positions

            IF current_coord IS NOT IN parent_map THEN
                RETURN NULL // Jalur terputus
            END IF
            SET current_coord TO parent_map[current_coord]
        END WHILE

        ADD new Position(x=start_coord[0], y=start_coord[1]) TO
path_positions
        REVERSE path_positions // Agar dari start ke target

        SET path_moves TO empty List
        FOR i FROM 0 TO LENGTH(path_positions) - 2
            SET p1 TO path_positions[i]
            SET p2 TO path_positions[i+1]
            ADD (p2.x - p1.x, p2.y - p1.y) TO path_moves
        END FOR
        RETURN path_moves
    END FUNCTION

    // Fungsi untuk menemukan diamond terbaik berdasarkan kriteria
(jarak atau utilitas)
    FUNCTION _find_best_diamond(diamonds: List of GameObject,
from_pos: Position, distance_map: Map, diamond_type: Optional String
= NULL, prioritize_distance_only: Boolean = FALSE) -> Optional
Position
        SET best_diamond_target TO NULL
        SET best_metric TO INFINITY IF prioritize_distance_only ELSE
-1.0

        FOR EACH diamond IN diamonds
            SET diamond_value TO diamond.properties.points IF
diamond.properties AND diamond.properties.points IS NOT NULL ELSE 1

```

```

        IF (diamond_type IS "red" AND diamond_value < 2) OR
(diamond_type IS "blue" AND diamond_value > 1) THEN
    CONTINUE // Lewati jika tipe tidak sesuai
END IF

        SET diamond_coord TO (diamond.position.x,
diamond.position.y)
        IF diamond_coord IS IN distance_map THEN
            SET distance TO distance_map[diamond_coord]
            IF distance IS 0 THEN CONTINUE // Jangan targetkan
diamond di posisi bot sendiri

            SET current_metric TO distance IF
prioritize_distance_only ELSE diamond_value / (distance + 0.000001)
            IF (prioritize_distance_only AND current_metric <
best_metric) OR (NOT prioritize_distance_only AND current_metric >
best_metric) THEN
                SET best_metric TO current_metric
                SET best_diamond_target TO diamond.position
            END IF
        END IF
    END FOR
    RETURN best_diamond_target
END FUNCTION

// Fungsi untuk menemukan sel terdekat yang belum dikunjungi
FUNCTION _find_nearest_unvisited(current_pos: Position, board:
Board, distance_map: Map) -> Optional Position
    SET nearest_unvisited_pos TO NULL
    SET min_dist_unvisited TO INFINITY
    FOR EACH cell_coord, dist IN distance_map.ITEMS()
        IF cell_coord IS NOT IN visited_cells THEN
            IF dist < min_dist_unvisited THEN
                SET min_dist_unvisited TO dist
                SET nearest_unvisited_pos TO new
Position(x=cell_coord[0], y=cell_coord[1])

```

```

        END IF
    END IF
END FOR
RETURN nearest_unvisited_pos
END FUNCTION

// Fungsi untuk menemukan diamond terdekat dari posisi awal tertentu (misalnya base)
FUNCTION _find_closest_diamond_from_start(diamonds: List of
GameObject, start_pos: Position, board: Board) -> Optional Position
SET base_distance_map, _ TO
calculate_all_distances(start_pos, board)
SET best_diamond_from_base TO NULL
SET min_dist_from_base TO INFINITY

FOR EACH diamond IN diamonds
    SET diamond_coord TO (diamond.position.x,
diamond.position.y)
    IF diamond_coord IS IN base_distance_map THEN
        SET distance_from_base TO
base_distance_map[diamond_coord]
        IF distance_from_base < min_dist_from_base THEN
            SET min_dist_from_base TO distance_from_base
            SET best_diamond_from_base TO diamond.position
        END IF
    END IF
END FOR
RETURN best_diamond_from_base
END FUNCTION

// Fungsi untuk menemukan red button terdekat
FUNCTION _find_red_button(board: Board, current_pos: Position,
distance_map: Map) -> Optional Position
SET red_buttons TO List of obj FROM board.game_objects WHERE
obj.type IS "artifact"
SET closest_button TO NULL

```

```

        SET min_dist TO INFINITY
        FOR EACH button IN red_buttons
            SET button_coord TO (button.position.x,
button.position.y)
            IF button_coord IS IN distance_map THEN
                SET dist TO distance_map[button_coord]
                IF dist < min_dist THEN
                    SET min_dist TO dist
                    SET closest_button TO button.position
                END IF
            END IF
        END FOR
        RETURN closest_button
    END FUNCTION

    // Fungsi untuk menemukan target tackle
    FUNCTION _find_tackle_target(board_bot: GameObject, board:
Board, distance_map: Map) -> Optional Position
        IF NOT board_bot.properties OR NOT
board_bot.properties.can_tackle THEN RETURN NULL
        SET best_tackle_target TO NULL
        SET max_diamonds_on_target TO -1
        SET all_other_bots TO List of bot FROM board.bots WHERE
bot.id IS NOT EQUAL TO board_bot.id

        FOR EACH other_bot IN all_other_bots
            SET other_bot_pos TO other_bot.position
            SET other_bot_coord TO (other_bot_pos.x,
other_bot_pos.y)
            IF other_bot_coord IS IN distance_map AND
distance_map[other_bot_coord] IS 1 THEN // Lawan di samping
                IF other_bot.properties AND
other_bot.properties.diamonds IS NOT NULL AND
other_bot.properties.diamonds >= TACKLE_DIAMOND_THRESHOLD THEN
                    IF other_bot.properties.diamonds >
max_diamonds_on_target THEN

```

```

        SET max_diamonds_on_target TO
other_bot.properties.diamonds

        SET best_tackle_target TO other_bot_pos
    END IF
END IF
END IF
END FOR

RETURN best_tackle_target
END FUNCTION

// Fungsi untuk menemukan tetangga teraman (terjauh dari bot
lawan)

FUNCTION _get_farthest_safe_neighbor(current_pos: Position,
board: Board, other_bots: List of GameObject) -> Optional
Tuple(Integer, Integer)

SET possible_moves TO [(0, 1), (0, -1), (1, 0), (-1, 0)]
SET best_move TO NULL
SET max_min_distance_to_enemy TO -1

FOR EACH dx, dy IN possible_moves
    SET next_x, next_y TO current_pos.x + dx, current_pos.y
+ dy
    SET next_pos TO new Position(x=next_x, y=next_y)

    IF board.is_valid_move(current_pos, dx, dy) THEN
        SET min_distance_to_enemy_for_this_move TO INFINITY
        FOR EACH other_bot IN other_bots
            SET dist_to_enemy TO ABS(next_pos.x -
other_bot.position.x) + ABS(next_pos.y - other_bot.position.y) // Jarak Manhattan
            SET min_distance_to_enemy_for_this_move TO
MIN(min_distance_to_enemy_for_this_move, dist_to_enemy)
        END FOR

        IF other_bots IS EMPTY OR
min_distance_to_enemy_for_this_move IS INFINITY THEN RETURN (dx, dy)
    END IF
END IF

```

```

        IF min_distance_to_enemy_for_this_move >
max_min_distance_to_enemy THEN
            SET max_min_distance_to_enemy TO
min_distance_to_enemy_for_this_move
            SET best_move TO (dx, dy)
        END IF
    END IF
END FOR

IF best_move IS NULL AND possible_moves IS NOT EMPTY THEN
    FOR EACH dx, dy IN possible_moves
        IF board.is_valid_move(current_pos, dx, dy) THEN
RETURN (dx, dy)
    END FOR
END IF
RETURN best_move
END FUNCTION

// Fungsi utama untuk menentukan gerakan bot berikutnya
FUNCTION next_move(board_bot: GameObject, board: Board) ->
Tuple(Integer, Integer)
    ADD (board_bot.position.x, board_bot.position.y) TO
visited_cells
    SET current_pos TO board_bot.position
    SET props TO board_bot.properties
    SET target_pos TO NULL

    SET distance_map_from_current, parent_map TO
_calculate_all_distances(current_pos, board)
    SET remaining_time_seconds TO props.milliseconds_left / 1000
IF props.milliseconds_left IS NOT NULL ELSE INFINITY
    SET other_bots TO List of bot FROM board.bots WHERE bot.id
IS NOT EQUAL TO board_bot.id

    // 1. Prioritas: Kembali ke base jika waktu kritis dan ada
diamond

```

```

    IF remaining_time_seconds < 10 AND props.diamonds IS NOT
NULL AND props.diamonds > 0 AND props.base IS NOT NULL THEN
        SET base_coord TO (props.base.x, props.base.y)
        IF base_coord IS IN distance_map_from_current THEN
            SET target_pos TO props.base
            // Jika target ditemukan, segera proses gerakan
            IF target_pos IS NOT NULL THEN
                SET path_moves TO
                _reconstruct_path_moves(current_pos, target_pos, parent_map)
                IF path_moves IS NOT NULL AND path_moves IS NOT
EMPTY THEN RETURN path_moves[0]
                ELSE // Fallback jika jalur tidak ditemukan
                    FOR EACH dx, dy IN [(1, 0), (-1, 0), (0, 1),
(0, -1)]
                        IF board.is_valid_move(current_pos, dx,
dy) THEN RETURN (dx, dy)
                    END FOR
                    RETURN (0, 0)
            END IF
            END IF
            END IF
        END IF
    END IF

    // 2. Prioritas: Menghindari bot lain jika membawa diamond
    IF target_pos IS NULL AND props.diamonds IS NOT NULL AND
props.diamonds > EVADE_DIAMOND_THRESHOLD AND other_bots IS NOT EMPTY
THEN
        SET should_evade TO FALSE
        FOR EACH other_bot IN other_bots
            SET dist_to_other_bot TO ABS(current_pos.x -
other_bot.position.x) + ABS(current_pos.y - other_bot.position.y)
            IF dist_to_other_bot <= EVADE_DETECTION_RANGE THEN
                SET should_evade TO TRUE
                BREAK
            END IF
        END FOR

```

```

        IF should_evade THEN
            SET safe_move TO
            _get_farthest_safe_neighbor(current_pos, board, other_bots)
            IF safe_move IS NOT NULL THEN RETURN safe_move // Langsung gerak menghindari
        END IF
    END IF

    // 3. Prioritas: Tackle bot lain
    IF target_pos IS NULL AND props.can_tackle AND
props.diamonds IS NOT NULL AND props.diamonds < 5 THEN
        SET tackle_target TO _find_tackle_target(board_bot,
board, distance_map_from_current)
        IF tackle_target IS NOT NULL THEN SET target_pos TO
tackle_target
    END IF

    // 4. Prioritas: Kembali ke base jika inventori penuh
    IF target_pos IS NULL AND props.diamonds IS NOT NULL AND
props.diamonds >= 5 AND props.base IS NOT NULL THEN
        SET base_coord TO (props.base.x, props.base.y)
        IF base_coord IS IN distance_map_from_current THEN SET
target_pos TO props.base
    END IF

    // 5. Prioritas: Kembali ke base / Cari diamond biru jika
inventori hampir penuh (4 diamond)
    IF target_pos IS NULL AND props.diamonds IS NOT NULL AND
props.diamonds IS 4 THEN
        SET target_pos TO _find_best_diamond(board.diamonds,
current_pos, distance_map_from_current, diamond_type="blue",
prioritize_distance_only=TRUE)
        IF target_pos IS NULL AND props.base IS NOT NULL THEN
            SET base_coord TO (props.base.x, props.base.y)
            IF (current_pos.x, current_pos.y) IS NOT EQUAL TO
base_coord AND base_coord IS IN distance_map_from_current THEN

```

```

        SET target_pos TO props.base
    END IF
END IF

// 6. Prioritas: Cari Red Button jika tidak ada diamond di
papan

IF target_pos IS NULL AND board.diamonds IS EMPTY THEN
    SET target_pos TO _find_red_button(board, current_pos,
distance_map_from_current)
END IF

// 7. Prioritas: Cari Diamond (Utilitas Terbaik / Jarak dari
Base)

IF target_pos IS NULL AND props.base IS NOT NULL THEN
    SET closest_diamond_from_base_pos TO
_find_closest_diamond_from_start(board.diamonds, props.base, board)
    IF closest_diamond_from_base_pos IS NOT NULL AND
(closest_diamond_from_base_pos.x, closest_diamond_from_base_pos.y)
IS IN distance_map_from_current THEN
        SET target_pos TO closest_diamond_from_base_pos
    END IF
END IF

IF target_pos IS NULL THEN // Jika belum ada target, cari
utilitas terbaik (merah dulu)

    SET target_pos TO _find_best_diamond(board.diamonds,
current_pos, distance_map_from_current, diamond_type="red",
prioritize_distance_only=FALSE)
END IF

IF target_pos IS NULL THEN // Kemudian biru
    SET target_pos TO _find_best_diamond(board.diamonds,
current_pos, distance_map_from_current, diamond_type="blue",
prioritize_distance_only=FALSE)
END IF

```

```

    // Fallback: Kembali ke base jika ada diamond tapi tidak ada
    target lain

        IF target_pos IS NULL AND props.diamonds IS NOT NULL AND
    props.diamonds > 0 AND props.base IS NOT NULL THEN
            SET base_coord TO (props.base.x, props.base.y)
            IF (current_pos.x, current_pos.y) IS NOT EQUAL TO
    base_coord AND base_coord IS IN distance_map_from_current THEN
                SET target_pos TO props.base
            END IF
        END IF

    // 8. Prioritas: Eksplorasi sel yang belum dikunjungi

    IF target_pos IS NULL THEN
        SET target_pos TO _find_nearest_unvisited(current_pos,
    board, distance_map_from_current)
    END IF

    // Eksekusi Gerakan Berdasarkan Target yang Ditemukan

    IF target_pos IS NOT NULL THEN
        SET current_target_position TO target_pos
        SET path_moves TO _reconstruct_path_moves(current_pos,
    target_pos, parent_map)
        IF path_moves IS NOT NULL AND path_moves IS NOT EMPTY
    THEN
            RETURN path_moves[0]
        ELSE // Jika jalur ke target utama tidak ditemukan
            SET current_target_position TO NULL
            // Coba eksplorasi terdekat sebagai fallback
            SET force_explore_target TO
    _find_nearest_unvisited(current_pos, board,
    distance_map_from_current)
            IF force_explore_target IS NOT NULL AND
    (force_explore_target.x, force_explore_target.y) IS NOT EQUAL TO
    (current_pos.x, current_pos.y) THEN

```

```

        SET path_to_explore TO
_reconstruct_path_moves(current_pos, force_explore_target,
parent_map)
        IF path_to_explore IS NOT NULL AND
path_to_explore IS NOT EMPTY THEN
            RETURN path_to_explore[0]
        END IF
    END IF
    END IF
END IF

// Fallback: Gerakan acak jika semua gagal
FOR EACH dx, dy IN [(1, 0), (-1, 0), (0, 1), (0, -1)]
    IF board.is_valid_move(current_pos, dx, dy) THEN RETURN
(dx, dy)
END FOR
RETURN (0, 0) // Tetap di tempat jika tidak ada gerakan yang
valid
END FUNCTION
END CLASS

```

4.1.2 Penjelasan Alur Program

Program bot bekerja dengan alur sebagai berikut:

1. Inisialisasi, yaitu bot menginisialisasi threshold dan struktur data yang diperlukan
2. Pembacaan State, yaitu bot membaca kondisi board melalui API game engine
3. Kalkulasi Jarak, dengan bot menghitung jarak ke semua posisi penting menggunakan BFS
4. Evaluasi Prioritas, dengan bot mengevaluasi 11 strategi Greedy berdasarkan prioritas dalam Waktu kritis → Return immediately Inventory penuh → Return to base Ancaman lawan → Escape atau tackle Target diamond → Pilih berdasarkan utilitas Fallback → Exploration atau random
5. Pemilihan Aksi, yaitu bot memilih strategi dengan prioritas tertinggi yang kondisinya terpenuhi
6. Eksekusi, yaitu bot mengirim aksi ke game engine
7. Update State, yaitu bot memperbarui visited cells dan posisi sebelumnya

4.2 Struktur Data yang Digunakan

1. Set (visited_cells), yaitu menyimpan koordinat sel yang sudah dikunjungi untuk strategi exploration.
2. Dictionary (distance_map), yaitu menyimpan jarak Manhattan dari posisi current ke semua posisi lain.
3. Tuple, yaitu menyimpan koordinat posisi (x, y).
4. List, yaitu menyimpan queue untuk BFS dalam perhitungan jarak.
5. Integer Constants, yaitu menyimpan threshold untuk berbagai strategi (EVADE_DIAMOND_THRESHOLD, TACKLE_DIAMOND_THRESHOLD).

4.3 Pengujian Program

4.3.1 Skenario Pengujian

Pengujian dilakukan dengan mensimulasikan berbagai kondisi permainan yang merepresentasikan kebutuhan dari tiap strategi greedy yang diterapkan. Setiap skenario mewakili situasi permainan yang khas.

Skenario 1: Waktu Kritis

- **Kondisi:** Waktu tersisa < 10 detik, bot memiliki 3 diamond, posisi dekat dengan base.
- **Tujuan:** Bot harus kembali ke base secepat mungkin.
- **Strategi Diuji:** Greedy by Return (Waktu Kritis)

Skenario 2: Ancaman Bot Lawan

- **Kondisi:** Bot memiliki 4 diamond, musuh mendekat dalam radius tackle, bot berada cukup jauh dari base.
- **Tujuan:** Bot harus menghindar untuk mempertahankan diamond.
- **Strategi Diuji:** Greedy by Escape

Skenario 3: Peluang Tackle

- **Kondisi:** Bot musuh memiliki 5 diamond dan berada dalam jangkauan tackle.
- **Tujuan:** Bot harus men-tackle musuh untuk mencuri diamond.
- **Strategi Diuji:** Greedy by Tackle

Skenario 4: Inventori Penuh

- **Kondisi:** Bot memiliki 5 diamond, belum dalam waktu kritis.
- **Tujuan:** Bot kembali ke base untuk menyetor diamond.
- **Strategi Diuji:** Greedy by Return (Inventori Penuh)

Skenario 5: Inventori Hampir Penuh

- **Kondisi:** Bot memiliki 4 diamond, 1 blue diamond terlihat dalam radius dekat.
- **Tujuan:** Ambil blue diamond atau langsung kembali ke base jika terlalu jauh.
- **Strategi Diuji:** Greedy by Return (Inventori Hampir Penuh)

Skenario 6: Tidak Ada Diamond Terlihat

- **Kondisi:** Seluruh papan tidak ada diamond aktif.
- **Tujuan:** Bot menuju red button terdekat untuk mengaktifkan diamond.
- **Strategi Diuji:** Greedy by Red Button

Skenario 7: Banyak Diamond Aktif

- **Kondisi:** Banyak diamond tersebar di peta (merah dan biru), tidak ada ancaman lawan.
- **Tujuan:** Bot memilih jalur dengan densitas diamond tertinggi yang bernilai tinggi.
- **Strategi Diuji:** Greedy by Density

Skenario 8: Area Tidak Dieksplorasi

- **Kondisi:** Bot tidak melihat diamond maupun red button di sekitar.
- **Tujuan:** Jelajahi area belum dikunjungi.
- **Strategi Diuji:** Greedy by Exploration

4.4 Hasil Pengujian dan Analisis

NO	Strategi Greedy Diuji	Keputusan Bot	Hasil
1	Greedy by Return (Waktu Kritis)	Kembali ke base	Diamond disetor tepat waktu

2	Greedy by Escape	Menjauh dari bot musuh ke zona aman	Diamond selamat
3	Greedy by Tackle	Men-tackle bot musuh	3 diamond dicuri
4	Greedy by Return (Inventori Penuh)	Menuju base terdekat	Point maksimal diperoleh
5	Greedy by Return (Inventori Hampir Penuh)	Mengambil 1 blue diamond dan ke base	Efisiensi waktu terjaga
6	Greedy by Red Button	Menuju dan mengaktifkan red button	Diamond muncul kembali
7	Greedy by Density	Menuju jalur dengan 2 red + 1 blue	Banyak diamond dikumpulkan
8	Greedy by Exploration	Menjelajah area yang belum terlihat	Menemukan 1 red diamond

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dalam permainan Diamonds, algoritma Greedy telah terbukti efektif dalam mencapai tujuan utama bot, yaitu mengumpulkan sebanyak mungkin diamond untuk memaksimalkan skor. Strategi Greedy yang dipilih yaitu, "8 Strategi dengan Sistem Prioritas Bertingkat," memungkinkan bot untuk membuat keputusan lokal yang ideal pada setiap langkah dengan mempertimbangkan berbagai faktor dinamis yang ada dalam permainan. Bot menavigasi board, memprioritaskan diamonds bernilai tinggi yaitu merah dibandingkan biru, dan secara proaktif mengelola stok dengan kembali ke rumah base saat diperlukan.

Selain itu, fleksibilitas dan peluang kemenangan bot telah ditingkatkan melalui penggabungan pertimbangan terhadap objek tertentu seperti red button dan teleporter, serta interaksi dengan bot lawan melalui menghindari atau melakukan tackle. Dalam konteks permainan Diamonds, yang memiliki sifat perubahan dinamis dan batasan waktu, pendekatan Greedy terbukti menjadi strategi yang layak dan kompetitif meskipun algoritma Greedy tidak selalu memberikan solusi optimal di seluruh dunia. Dalam lingkungan permainan real-time, kemampuan bot untuk membuat keputusan cepat berdasarkan kondisi sesaat sangat penting.

5.2 Saran

Untuk pengembangan lebih lanjut, disarankan:

1. Implementasi Algoritma Pathfinding, dengan menggunakan algoritma untuk path planning yang lebih optimal dibandingkan greedy movement.
2. Prediksi Pergerakan Lawan, yaitu dengan menambahkan machine learning atau pattern recognition untuk memprediksi pergerakan bot lawan.
3. Optimasi Threshold, dengan melakukan tuning parameter threshold berdasarkan analisis statistik dari berbagai skenario permainan.
4. Strategi Kooperatif, yaitu jika permainan mendukung multiple bot dalam satu tim, implementasikan strategi koordinasi antar bot.
5. Dynamic Risk Assessment, yaitu dengan mengembangkan sistem penilaian risiko yang lebih sophisticated untuk keputusan tackle dan escape.

6. Memory-based Learning, dengan menyimpan informasi tentang pola permainan untuk pembelajaran adaptif dalam jangka panjang.

LAMPIRAN

A. Repository Github

https://github.com/Nadya-167-14/Stima_DiamondBot_tripleN/tree/main

B. Link GDrive

[https://drive.google.com/drive/folders/15whZ8HNK-4OpT18ncR7cvLyKbnqb9QPJ
?usp=sharing](https://drive.google.com/drive/folders/15whZ8HNK-4OpT18ncR7cvLyKbnqb9QPJ?usp=sharing)

DAFTAR PUSTAKA

- [1] *N. F. Lakutu, M. R. Katili, S. L. Mahmud and I. N. Yahya, "Algoritma Dijkstra dan Algoritma Greedy Untuk Optimasi Rute," Euler: Jurnal Ilmiah Matematika, Sains dan Teknologi, vol. 1, no. 11, pp. 55-65, 2023.*
- [2] *Y. Darnita and R. Toyib, "Penerapan Algoritma Greedy Dalam Pencarian Jalur Terpendek Pada Instansi-Instansi Penting Di Kota Argamakmur Kabupaten Bengkulu Utara," Jurnal Media Infotama, vol. 2, no. 15, 2019.*
- [3] *R. Munir, "Algoritma Greedy," Departemen Teknik Informatika Institut Teknologi Bandung, Bandung, 2004.*