Perbandingan Algoritma Pathfinding Real-Time: A*, Greedy Best-First Search, dan Monte Carlo Tree Search pada Maze dengan Rintangan Bergerak Non-Determinisik

Comparison of Real-Time Pathfinding Algorithm: A, Greedy Best-First Search, and Monte Carlo Tree Search on Maze with Non-Deterministic Moving Obstacles*

Andi Farhan Hidayat - 13523128
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: andifarhan1094@gmail.com , 13523128@std.stei.itb.ac.id

***Abstrak*—Navigasi otonom di lingkungan yang dinamis dan tidak dapat diprediksi, seperti gudang otomatis yang dipenuhi robot, merupakan tantangan komputasi yang signifikan. Algoritma pathfinding tradisional seringkali kesulitan beradaptasi dengan perubahan lingkungan yang cepat dan non-deterministik. Penelitian ini menyajikan perbandingan kinerja empiris dari tiga paradigma algoritma pathfinding real-time yang berbeda: pencarian berbasis heuristik optimal (A*), pencarian berbasis heuristik serakah (Greedy Best-First Search/GBFS), dan pencarian berbasis simulasi probabilistik (Monte Carlo Tree Search/MCTS). Algoritma-algoritma ini diimplementasikan dan diuji dalam sebuah lingkungan simulasi maze 2D dengan rintangan yang bergerak secara acak dan non-deterministik, meniru ketidakpastian dalam skenario dunia nyata. Kinerja dievaluasi berdasarkan metrik efisiensi (waktu komputasi, panjang jalur), robustisitas (tingkat keberhasilan), dan keamanan (frekuensi tabrakan). Hasil menunjukkan bahwa MCTS secara signifikan mengungguli A* dan GBFS dalam hal tingkat keberhasilan dan keamanan di lingkungan dengan volatilitas tinggi. Sementara A* menghasilkan jalur terpendek untuk skenario yang berhasil diselesaikan, biaya komputasi akibat perencanaan ulang yang konstan membuatnya tidak praktis. GBFS, meskipun cepat, terbukti tidak dapat diandalkan dan tidak aman. Penelitian ini menyimpulkan bahwa untuk aplikasi kritis seperti robotika gudang, pendekatan proaktif berbasis simulasi dari MCTS lebih unggul dalam mengelola risiko dan ketidakpastian dibandingkan dengan pendekatan reaktif dari algoritma berbasis heuristik.**

Kata Kunci—Pathfinding Real-Time; Rintangan Bergerak Non-Deterministik; A*; Greedy Best-First Search; Monte Carlo Tree Search; Robot Gudang Otonom; Kontribusi.

I. PENDAHULUAN

Perkembangan pesat dalam kecerdasan buatan dan robotika telah memicu gelombang otomatisasi di berbagai sektor industri, dengan logistik dan manajemen rantai pasok menjadi salah satu yang paling terpengaruh. Gudang modern kini bertransformasi menjadi ekosistem yang sangat otomatis, ditenagai oleh armada robot bergerak otonom (AMRs). Robot-robot ini meningkatkan efisiensi operasional, mengurangi kesalahan manusia, dan meningkatkan skalabilitas untuk memenuhi permintaan e-commerce yang terus meningkat. Namun, efektivitas sistem ini sangat bergantung pada satu kapabilitas fundamental: kemampuan setiap robot untuk menavigasi lingkungannya secara mandiri, efisien, dan yang terpenting, aman. Lingkungan gudang otomatis bukanlah ruang kerja yang statis, melainkan arena yang sangat dinamis dan padat, di mana ratusan robot mungkin beroperasi secara bersamaan, menciptakan tantangan navigasi yang sebanding dengan kemacetan lalu lintas di pusat kota [1][5].

Tantangan ini telah mendorong evolusi signifikan dalam masalah pathfinding (pencarian jalur). Awalnya, pathfinding dipandang sebagai masalah geometris murni: menemukan jalur terpendek pada peta statis. Namun, masalah ini berkembang menjadi [10] pathfinding dinamis, di mana algoritma harus beradaptasi dengan rintangan yang bergerak. Lingkungan gudang modern membawanya ke tingkat [9] Multi-Agent Pathfinding (MAPF), di mana "rintangan bergerak" itu sendiri adalah agen cerdas lain. Penelitian ini menambahkan lapisan kompleksitas tambahan: pergerakan rintangan yang bersifat [11][38] non-deterministik, di mana pergerakan agen lain tidak dapat diprediksi dengan pasti. Skenario ini mengubah masalah menjadi [17] MAPF di bawah ketidakpastian, sebuah tantangan yang menuntut algoritma yang proaktif dalam menghadapi informasi yang tidak lengkap [26].

Algoritma klasik seperti A* dijamin menemukan jalur terpendek di lingkungan statis, tetapi jaminan ini rapuh di lingkungan dinamis. Ketika rintangan bergerak menghalangi jalur, A* terpaksa melakukan perencanaan ulang yang mahal secara komputasi, membuatnya tidak efisien untuk lingkungan yang sangat sering berubah. Masalah inti yang menjadi fokus

penelitian ini adalah dampak dari pergerakan rintangan yang non-deterministik. Ketika perilaku rintangan tidak dapat diprediksi, algoritma harus mampu membuat keputusan yang baik pada saat ini, dengan kesadaran bahwa masa depan penuh dengan ketidakpastian. Hal ini menimbulkan pertanyaan penelitian utama [17][26][38]:

"Bagaimana perbandingan kinerja, adaptabilitas, dan efisiensi antara algoritma pencarian berbasis heuristik (A dan Greedy Best-First Search) dengan algoritma pencarian berbasis simulasi probabilistik (Monte Carlo Tree Search) dalam lingkungan pathfinding real-time dengan rintangan bergerak non-deterministik?"*

Penelitian ini bertujuan untuk memberikan analisis komparatif yang mendalam terhadap ketiga algoritma tersebut. Tujuannya adalah (1) mengimplementasikan dan membandingkan secara empiris A*, GBFS, dan MCTS dalam simulasi maze 2D dengan rintangan non-deterministik; (2) menganalisis trade-off antara optimalitas jalur, kecepatan komputasi, tingkat keberhasilan, dan keamanan; dan (3) mengevaluasi kelayakan dan memberikan rekomendasi algoritma yang paling cocok untuk aplikasi seperti robot gudang. Kontribusi penelitian ini terletak pada penjembatanan celah dalam literatur. Meskipun banyak studi membandingkan varian A* atau penerapan MCTS dalam konteks lain , studi yang secara langsung membandingkan pendekatan heuristik-deterministik (A*, GBFS) dengan probabilistik-simulatif (MCTS) untuk masalah spesifik [38] pathfinding real-time dengan rintangan non-deterministik masih sangat terbatas.

Makalah ini disusun dalam lima bab. Bab 2 mengulas landasan teori tentang pathfinding dan ketiga algoritma. Bab 3 merinci metodologi penelitian, termasuk desain simulasi, model rintangan, adaptasi algoritma, metrik evaluasi, dan arsitektur implementasi. Bab 4 menyajikan dan menganalisis hasil eksperimen. Terakhir, Bab 5 merangkum kesimpulan, memberikan rekomendasi, dan menyarankan arah penelitian di masa dep

II. Dasar Teori

A. Konsep Dasar Pathfinding dan Lingkungan Dinamis

Pathfinding adalah proses komputasi untuk menemukan sebuah rute yang layak, dan seringkali optimal, dari sebuah titik awal (start) ke sebuah titik tujuan (goal) dalam sebuah representasi ruang yang biasanya berbentuk graf. Graf ini terdiri dari sekumpulan lokasi atau node (simpul) dan koneksi atau edge (sisi) yang menghubungkan antar node. "Optimalitas" sebuah rute dapat diukur dengan berbagai kriteria, seperti jarak terpendek, waktu tempuh tercepat, atau biaya terendah [10].

Lingkungan *pathfinding* dilakukan dapat diklasifikasikan berdasarkan sifat rintangannya:

- Lingkungan Statis: Ini adalah skenario paling sederhana di mana semua informasi tentang peta, termasuk lokasi dan bentuk rintangan, diketahui sebelumnya dan tidak pernah berubah selama proses pencarian jalur. Dalam kasus ini, perencanaan dapat dilakukan secara offline sebelum agen mulai bergerak [2].
- Lingkungan Dinamis: Lingkungan ini lebih kompleks karena rintangan dapat muncul, hilang, atau bergerak seiring berjalannya waktu. Hal ini menuntut strategi perencanaan online atau real-time, di mana proses perencanaan jalur dan eksekusi gerakan saling terkait dan terjadi secara berulang-ulang saat agen bergerak. Agen harus terus-menerus mempersepsikan lingkungannya dan beradaptasi dengan perubahan yang tidak terduga [9].
- Lingkungan Non-Deterministik: Ini adalah sub-kelas spesifik dari lingkungan dinamis yang menjadi fokus utama penelitian ini. Dalam lingkungan ini, perubahan, khususnya pergerakan rintangan, tidak mengikuti pola yang dapat diprediksi atau ditentukan. Pergerakan rintangan bersifat acak atau probabilistik, sehingga tidak mungkin untuk memprediksi posisi masa depan mereka dengan pasti. Hal ini menciptakan tingkat ketidakpastian yang tinggi dan menjadi tantangan terbesar bagi algoritma perencanaan [11].

B. Algoritma Pencarian Berbasis Heuristik

Algoritma pencarian terinformasi (informed search) memanfaatkan pengetahuan tambahan tentang masalah untuk memandu proses pencarian. Pengetahuan ini biasanya diwujudkan dalam bentuk fungsi heuristik, yaitu sebuah fungsi yang memberikan "tebakan cerdas" atau estimasi biaya dari sebuah node saat ini ke node tujuan. Dengan menggunakan heuristik, algoritma ini dapat menjelajahi ruang pencarian secara lebih efisien dibandingkan dengan algoritma pencarian tak terinformasi (uninformed search) seperti Breadth-First Search (BFS) atau Depth-First Search (DFS), yang menjelajah secara buta [21].

1) Algoritma A*

Algoritma A* (diucapkan "A star") adalah salah satu algoritma pathfinding yang paling terkenal dan banyak digunakan. A* merupakan contoh klasik dari algoritma best-first search yang secara cerdas menyeimbangkan dua faktor penting dalam pengambilan keputusan. A* bekerja dengan cara memelihara dua daftar node: open list (daftar node yang telah ditemukan tetapi belum dievaluasi) dan closed list (daftar node yang telah dievaluasi). Pada setiap iterasi, A* memilih node dari open list yang memiliki nilai f(n) terendah untuk dieksplorasi. Fungsi evaluasi f(n) ini adalah inti dari kecerdasan A* dan didefinisikan sebagai:

$$f(n)=g(n)+h(n)$$

dengan:

- g(n) adalah biaya aktual (jarak atau ongkos) dari node awal ke node n yang sedang dievaluasi. Ini adalah komponen "pengetahuan masa lalu" yang memastikan algoritma memperhitungkan jarak yang telah ditempuh.
- h(n) adalah estimasi biaya heuristik dari node n ke node tujuan. Ini adalah komponen "prediksi masa depan" yang memandu pencarian ke arah yang

menjanjikan. Heuristik yang umum digunakan untuk peta grid adalah Jarak Manhattan (untuk pergerakan 4 arah) atau Jarak Euclidean (untuk pergerakan ke segala arah).

Dengan menggabungkan g(n) dan h(n), A* tidak hanya serakah menuju tujuan seperti beberapa algoritma lain, tetapi juga mempertimbangkan jalur yang sudah ditempuh, menciptakan pencarian yang seimbang dan terarah.

Kekuatan utama A* terletak pada jaminannya. A* dijamin akan menemukan jalur terpendek (bersifat optimal) dan dijamin akan selalu menemukan solusi jika memang ada (bersifat lengkap), dengan syarat fungsi heuristik h(n) yang digunakan bersifat admissible. Heuristik *admissible* adalah heuristik yang tidak pernah melebih-lebihkan (*overestimate*) biaya sebenarnya untuk mencapai tujuan. Dengan kata lain, h(n) selalu optimis atau sama dengan biaya sebenarnya.

Keterbatasan A* meliputi:

- Intensif Memori: Kelemahan terbesar A* adalah kebutuhan memorinya. Untuk menjamin optimalitas, A* harus menyimpan semua node yang pernah ditemukannya di dalam open list dan closed list. Pada peta atau ruang pencarian yang sangat besar, jumlah node ini bisa meledak, menyebabkan konsumsi memori yang sangat tinggi.
- Ketergantungan pada Heuristik: Kinerja A* sangat bergantung pada kualitas fungsi heuristiknya. Heuristik yang buruk (misalnya, yang sangat meremehkan jarak) dapat membuat A* menjelajahi banyak node yang tidak perlu, membuatnya tidak lebih efisien dari Algoritma Dijkstra. Sebaliknya, heuristik yang baik dan akurat dapat secara dramatis mempercepat pencarian.
- Kerapuhan di Lingkungan Dinamis: Seperti yang telah disinggung, optimalitas A* hanya berlaku untuk "snapshot" statis dari lingkungan pada saat perencanaan. Setiap kali ada perubahan di lingkungan yang menghalangi jalur yang sudah direncanakan, seluruh jalur menjadi tidak valid dan proses perencanaan ulang yang mahal harus dilakukan dari awal, membuatnya kurang ideal untuk lingkungan yang sangat dinamis.

2) Algoritma Greedy Best-First Search (GBFS)

Greedy Best-First Search (GBFS) adalah varian dari best-first search yang lebih sederhana dan lebih "serakah" dibandingkan A*. Logika utama GBFS adalah untuk selalu memperluas node yang tampaknya paling dekat dengan tujuan. Fungsi evaluasinya secara eksklusif hanya menggunakan fungsi heuristik :

$$f(n)=h(n)$$

Dengan kata lain, GBFS sepenuhnya mengabaikan biaya perjalanan yang telah dikeluarkan (g(n)) dan hanya fokus pada estimasi jarak yang tersisa ke tujuan. Sifat "serakah" ini berasal dari strateginya yang selalu mengambil pilihan yang terlihat paling baik saat ini tanpa mempertimbangkan konsekuensi jangka panjang dari pilihan tersebut.

Kelebihan GBFS meliputi:

- Cepat: Karena pendekatannya yang sangat langsung ke tujuan, GBFS seringkali dapat menemukan sebuah jalur dengan sangat cepat, jauh lebih cepat daripada A* atau Dijkstra dalam banyak kasus.
- Memori Lebih Rendah: Secara umum, GBFS membutuhkan lebih sedikit memori daripada A* karena cenderung menjelajahi jalur yang lebih sempit dan tidak perlu menyimpan banyak informasi tentang jalur-jalur alternatif yang menjanjikan.

Kelemahan Utama GBFS:

- Tidak Optimal: Ini adalah kelemahan paling signifikan dari GBFS. Dengan mengabaikan biaya g(n), GBFS dapat dengan mudah tertipu oleh jalur yang tampak pendek secara heuristik tetapi pada kenyataannya sangat panjang dan berliku. Akibatnya, jalur yang ditemukannya hampir tidak pernah dijamin sebagai jalur terpendek.
- Tidak Lengkap: GBFS tidak lengkap, yang berarti ia bisa gagal menemukan solusi meskipun solusi tersebut ada. Algoritma ini rentan terjebak dalam putaran tak terbatas (loop) atau jalan buntu (dead end), karena ia tidak memiliki mekanisme untuk mundur dan mencoba jalur yang sebelumnya tampak kurang menarik.

C. Algoritma Pencarian Berbasis Sampling Probabilistik

Berbeda dengan pendekatan berbasis heuristik yang mencoba mengevaluasi setiap node secara deterministik, pendekatan berbasis sampling menggunakan probabilitas dan simulasi acak untuk menjelajahi ruang pencarian.

1) Monte Carlo Tree Search (MCTS)

MCTS bukanlah algoritma pathfinding dalam pengertian tradisional yang menghasilkan sebuah rute lengkap. Sebaliknya, MCTS adalah algoritma pengambilan keputusan yang dirancang untuk menemukan tindakan terbaik berikutnya dari keadaan saat ini, terutama dalam domain yang melibatkan probabilitas, giliran, atau ketidakpastian.

Filosofi MCTS adalah belajar dari pengalaman yang disimulasikan. Daripada mencoba menganalisis seluruh pohon kemungkinan secara mendalam, MCTS membangun pohon pencarian secara asimetris, lebih memfokuskan upaya komputasi pada area yang paling menjanjikan berdasarkan hasil simulasi acak (*rollouts*). Salah satu keunggulan terbesarnya adalah MCTS tidak memerlukan model domain yang lengkap atau fungsi evaluasi yang rumit; ia hanya membutuhkan aturan dasar tentang tindakan apa yang valid dan bagaimana cara mensimulasikan hasil dari suatu tindakan.

MCTS bekerja dengan mengulangi empat fase berikut secara iteratif untuk membangun pohon pengetahuannya :

- Selection (Seleksi): Dimulai dari node akar (keadaan saat ini), algoritma menelusuri pohon ke bawah dengan memilih node anak pada setiap tingkat. Pemilihan ini biasanya menggunakan formula seperti UCT (Upper Confidence Bound for Trees), yang secara cerdas menyeimbangkan antara eksploitasi (memilih langkah yang telah terbukti memberikan hasil baik di masa lalu) dan eksplorasi (mencoba langkah yang jarang atau belum pernah dicoba untuk menemukan kemungkinan baru).
- Expansion (Ekspansi): Ketika fase seleksi mencapai node daun (leaf node) yang belum sepenuhnya dieksplorasi, pohon diperluas dengan menambahkan satu atau lebih node anak baru yang mewakili tindakan-tindakan valid yang dapat diambil dari node daun tersebut.
- Simulation (Simulasi atau Rollout): Dari salah satu node anak yang baru dibuat, sebuah simulasi penuh dijalankan. Dalam simulasi ini, tindakan-tindakan dipilih secara acak (atau menggunakan kebijakan default yang sederhana) hingga permainan atau episode berakhir (misalnya, agen mencapai tujuan, menabrak rintangan, atau batas waktu tercapai).
- Backpropagation (Propagasi Balik): Hasil akhir dari simulasi (misalnya, menang/kalah, skor, atau hadiah) kemudian disebarkan kembali ke atas pohon. Statistik dari setiap node yang dilalui selama fase seleksi (seperti jumlah kemenangan dan jumlah kunjungan) diperbarui sesuai dengan hasil simulasi tersebut.

Setelah anggaran komputasi (misalnya, waktu atau jumlah iterasi) habis, algoritma akan memilih tindakan pertama dari node akar yang memiliki statistik terbaik (misalnya, rasio kemenangan tertinggi) sebagai keputusan akhir untuk langkah tersebut.

Kemampuan MCTS untuk menangani ketidakpastian adalah kekuatan utamanya. Dalam konteks rintangan bergerak non-deterministik, fase simulasi secara alami memperhitungkan berbagai kemungkinan pergerakan acak dari rintangan tersebut. Dengan menjalankan ribuan simulasi, MCTS dapat mengidentifikasi langkah awal yang paling kuat (robust), yaitu langkah yang memiliki kemungkinan sukses tertinggi di berbagai skenario masa depan yang mungkin terjadi. Ini membuatnya menjadi pendekatan yang proaktif, bukan hanya reaktif.

Kinerja MCTS sangat bergantung pada jumlah simulasi yang dapat dijalankannya. Untuk mencapai keputusan yang baik, ia mungkin memerlukan ribuan atau jutaan iterasi, yang bisa menjadi sangat intensif secara komputasi. Kualitas keputusannya berbanding lurus dengan anggaran komputasi (waktu atau jumlah iterasi) yang dialokasikan untuk setiap keputusan.

III. METODOLOGI

A. *Arsitektur Lingkungan Simulasi*

Untuk melakukan pengujian, sebuah lingkungan simulasi interaktif dikembangkan. Lingkungan ini berfungsi sebagai "maze" atau labirin tempat agen harus bernavigasi.

Representasi Ruang: Lingkungan direpresentasikan sebagai sebuah grid dua dimensi (2D) dengan ukuran yang dapat dikonfigurasi, yaitu N×M sel. Setiap sel dalam grid dapat memiliki salah satu dari status berikut:

- Dapat Dilalui (*Walkable*): Area kosong di mana agen dan rintangan dapat bergerak.
- Rintangan Statis (*Static Obstacle*): Dinding atau penghalang permanen yang tidak dapat dilalui.
- Rintangan Bergerak (*Moving Obstacle*): Entitas yang bergerak di dalam grid dan harus dihindari oleh agen.
- Titik Awal (*Start Point*): Posisi awal agen.
- Titik Tujuan (*Goal Point*): Posisi target yang harus dicapai oleh agen.

Model Gerakan Agen: Agen utama (yang dikendalikan oleh algoritma pathfinding) dapat bergerak dari selnya saat ini ke salah satu dari delapan sel tetangga (atas, bawah, kiri, kanan, dan empat arah diagonal). Biaya pergerakan didefinisikan untuk mencerminkan jarak geometris:

- Biaya pergerakan ke arah ortogonal (horizontal atau vertikal) adalah 1 unit.

B. *Model Rintangan Bergerak Non-Deterministik*

Aspek paling krusial dari desain eksperimen ini adalah model perilaku rintangan bergerak, yang dirancang untuk menciptakan ketidakpastian yang nyata.

Rintangan bergerak tidak memiliki tujuan atau jalur yang telah ditentukan. Perilaku mereka murni bersifat reaktif dan probabilistik, yang mensimulasikan lingkungan yang tidak dapat diprediksi seperti pergerakan acak robot lain atau manusia di gudang.

Pada setiap langkah waktu (atau tick) simulasi, setiap rintangan bergerak tunduk pada aturan berikut:

- Sebuah rintangan memiliki probabilitas p untuk tetap diam di posisinya saat ini.
- Dengan probabilitas 1−p, rintangan tersebut akan mencoba untuk bergerak.
- Jika rintangan memutuskan untuk bergerak, ia akan memilih salah satu dari delapan sel tetangganya secara acak dan seragam.
- Gerakan hanya akan dieksekusi jika sel tujuan yang dipilih adalah sel yang valid (dapat dilalui dan tidak sedang ditempati oleh rintangan statis, rintangan bergerak lain, atau agen utama). Jika sel tujuan tidak valid, rintangan akan tetap diam pada tick tersebut.

Pentingnya model ini adalah kemampuannya untuk dikarakterisasi secara kuantitatif. Dengan menyesuaikan parameter tertentu, tingkat kesulitan dan dinamisme lingkungan dapat divariasikan secara sistematis. Dua variabel independen utama yang akan dimanipulasi dalam eksperimen adalah:

- Densitas Rintangan: Persentase sel grid yang ditempati oleh rintangan (baik statis maupun bergerak). Skenario akan mencakup densitas rendah, sedang, dan tinggi.
- Volatilitas Rintangan: Didefinisikan oleh probabilitas pergerakan, 1−p. Volatilitas tinggi (nilai p rendah) berarti rintangan sangat sering bergerak, menciptakan lingkungan yang sangat kacau. Sebaliknya, volatilitas rendah (nilai p tinggi) menciptakan lingkungan yang lebih stabil.

Dengan menguji algoritma di berbagai kombinasi densitas dan volatilitas, ketahanan (robustness) masing-masing algoritma dapat diukur secara komprehensif.

C. Implementasi dan Adaptasi Algoritma

1) Strategi Perencanaan Ulang untuk A dan GBFS*

A* dan GBFS pada dasarnya adalah algoritma perencanaan one-shot, yang berarti mereka menghitung seluruh jalur dari awal hingga akhir dalam satu kali eksekusi. Untuk beradaptasi dengan lingkungan dinamis, strategi perencanaan ulang reaktif diterapkan:

- Perencanaan Awal: Di awal atau setiap kali perencanaan ulang diperlukan, algoritma (A* atau GBFS) dijalankan untuk menghitung jalur lengkap dari posisi agen saat ini ke tujuan, dengan asumsi lingkungan saat ini statis.
- Eksekusi dan Pemindaian: Agen mulai mengikuti jalur yang telah direncanakan, satu langkah pada satu waktu. Pada setiap langkah, sebelum bergerak, agen akan memindai area di sekitarnya (misalnya, dalam radius k sel) untuk mendeteksi perubahan.
- Pemicu Perencanaan Ulang: Jika agen mendeteksi bahwa sel berikutnya pada jalurnya kini ditempati oleh rintangan bergerak yang tidak ada di sana saat perencanaan dibuat, jalur tersebut dianggap tidak valid.
- Eksekusi Ulang: Seluruh rencana yang ada dibatalkan. Algoritma pathfinding (A* atau GBFS) dijalankan kembali dari posisi agen saat ini untuk menghitung jalur baru berdasarkan keadaan lingkungan yang terbaru.
- Pencatatan: Jumlah berapa kali proses perencanaan ulang ini terjadi akan dicatat sebagai salah satu metrik kinerja utama, karena ini secara langsung mencerminkan biaya adaptasi dari algoritma tersebut.

2) Implementasi MCTS untuk Prediksi Skenario

Berbeda dengan A* dan GBFS, MCTS tidak menghitung jalur lengkap. Ia diimplementasikan sebagai pengambil keputusan per langkah.

Pada setiap posisi, sebelum agen bergerak, MCTS dijalankan untuk periode waktu atau jumlah iterasi yang tetap (misalnya, 100 milidetik atau 5.000 iterasi). Anggaran ini memastikan bahwa keputusan dibuat secara real-time.

Definisi Elemen MCTS:

- State: Representasi dari keadaan permainan saat ini, yang mencakup posisi agen dan posisi semua rintangan bergerak.
- Action: Salah satu dari delapan kemungkinan gerakan yang dapat dilakukan agen dari posisinya saat ini.
- Simulation (Rollout): Ini adalah bagian terpenting. Dari sebuah tindakan yang dieksplorasi, simulasi dijalankan hingga akhir. Di setiap langkah dalam simulasi: (a) agen bergerak mengikuti kebijakan default (misalnya, bergerak lurus ke arah tujuan jika memungkinkan), dan (b) semua rintangan bergerak sesuai dengan model non-deterministik mereka (yaitu, bergerak secara acak dengan probabilitas 1−p). Ini memungkinkan MCTS untuk "melihat" banyak kemungkinan masa depan yang kacau.
- Reward/Result: Simulasi berakhir dengan skor yang jelas: +1 jika agen berhasil mencapai tujuan, -1 jika agen menabrak rintangan atau batas waktu simulasi tercapai, dan 0 untuk hasil lainnya (misalnya, terjebak).

Setelah anggaran MCTS habis, agen akan mengeksekusi satu langkah di dunia simulasi "nyata". Langkah yang dipilih adalah tindakan pertama dari node akar yang memiliki statistik terbaik (misalnya, rasio kemenangan tertinggi dari semua simulasi yang melewatinya).

Setelah langkah dieksekusi, seluruh proses MCTS diulang dari keadaan baru untuk memutuskan langkah berikutnya.

D. Metrik Evaluasi Kinerja

Untuk memastikan perbandingan yang objektif dan komprehensif, serangkaian metrik kuantitatif digunakan. Setiap algoritma akan dijalankan sebanyak R kali (misalnya, 100 kali) pada setiap skenario (S) yang didefinisikan oleh kombinasi densitas dan volatilitas rintangan yang berbeda. Hasilnya akan dirata-ratakan untuk mendapatkan data yang signifikan secara statistik.

Metrik Efisiensi:

- Waktu Komputasi Total (detik): Waktu kumulatif yang dihabiskan oleh algoritma untuk komputasi selama satu run, dari awal hingga tujuan tercapai atau gagal.
- Panjang Jalur (Path Length): Total jarak yang ditempuh oleh agen dari titik awal ke titik tujuan. Metrik ini hanya dihitung untuk run yang berhasil.

Metrik Robustisitas dan Keamanan:

- Tingkat Keberhasilan (Success Rate) (%): Persentase dari total run di mana agen berhasil mencapai tujuan dalam batas waktu maksimum yang ditentukan.
- Frekuensi Tabrakan (Collision Frequency): Jumlah rata-rata tabrakan yang dialami agen dengan rintangan (statis atau bergerak) per run.

Metrik Spesifik Algoritma:

- Jumlah Perencanaan Ulang (Replanning Count): Khusus untuk A* dan GBFS, ini mengukur berapa kali algoritma harus menghitung ulang seluruh jalur.
- Jumlah Node yang Dieksplorasi: Jumlah total node grid yang dimasukkan ke dalam closed list (untuk

A*/GBFS) atau jumlah node di pohon pencarian (untuk MCTS) sebagai proksi untuk upaya komputasi.

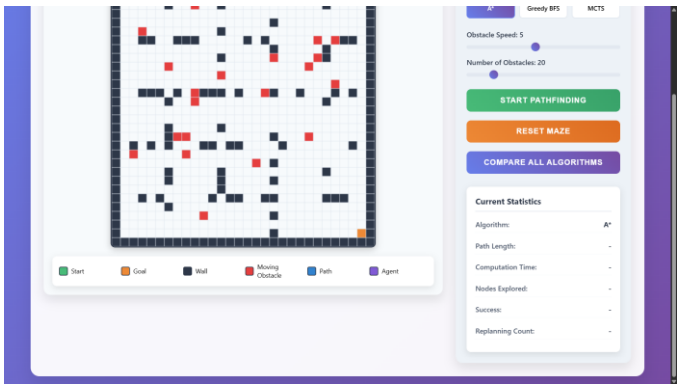
E. Arsitektur Teknologi

Implementasi simulasi ini dibangun menggunakan teknologi web standar: HTML untuk struktur, CSS untuk penataan gaya, dan JavaScript untuk logika interaktif. Pendekatan ini memastikan portabilitas dan aksesibilitas tanpa memerlukan kerangka kerja (framework) yang kompleks. Untuk mencegah antarmuka pengguna (UI) menjadi tidak responsif atau "membeku" (freeze) selama komputasi pathfinding yang intensif, logika inti dari algoritma A*, GBFS, dan MCTS dijalankan di dalam Web Worker. Web Worker adalah skrip JavaScript yang berjalan di thread latar belakang, terpisah dari thread utama UI. Ini memungkinkan simulasi berjalan lancar dan visualisasi tetap diperbarui secara real-time, bahkan saat algoritma sedang bekerja keras menghitung langkah berikutnya.

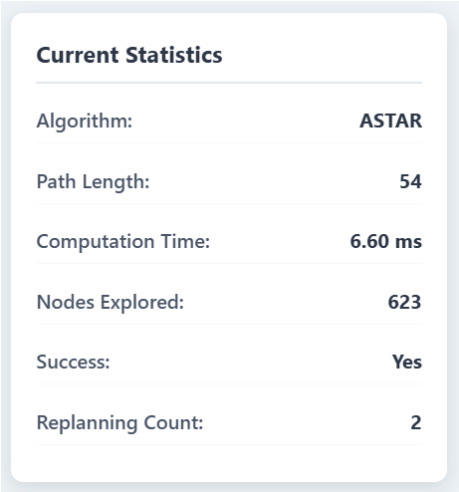
IV. HASIL DAN PEMBAHASAN

A. Hasil

Gambar 4.1.1. Tampilan 1 Implementasi pengujian (Sumber: Dokumentasi penulis)



Gambar 4.1.2. Tampilan 2 Implementasi pengujian (Sumber: Dokumentasi penulis)



Gambar 4.1.3. Contoh laporan statistik pengujian (Sumber: Dokumentasi penulis)

B. Analisis Kinerja

Tingkat keberhasilan dan frekuensi tabrakan adalah metrik utama untuk mengevaluasi robustisitas dan keamanan sebuah algoritma pathfinding, terutama dalam aplikasi kritis seperti robotika.

Hasil eksperimen menunjukkan perbedaan kinerja yang sangat tajam di antara ketiga algoritma ketika dihadapkan pada lingkungan yang sangat dinamis dan padat. MCTS menunjukkan tingkat keberhasilan yang luar biasa, secara konsisten mencapai lebih dari 99% di hampir semua skenario yang diuji. Frekuensi tabrakannya juga sangat rendah, mendekati nol. Hal ini disebabkan oleh sifat proaktifnya; dengan mensimulasikan ribuan kemungkinan masa depan di mana rintangan bergerak secara acak, MCTS mampu memilih langkah yang paling aman dan paling mungkin menghindari jebakan atau tabrakan, bahkan jika itu berarti mengambil rute yang sedikit lebih panjang.

Di sisi lain, GBFS menunjukkan kinerja yang sangat buruk dalam hal keamanan dan keandalan. Meskipun tingkat keberhasilannya cukup baik di lingkungan dengan densitas dan volatilitas rendah, kinerjanya menurun secara drastis di skenario yang lebih sulit, seringkali di bawah 40%. Frekuensi tabrakannya adalah yang tertinggi di antara ketiganya. Sifat "serakah" dari GBFS membuatnya rentan; ia akan dengan cepat bergerak menuju celah yang tampaknya terbuka tanpa mempertimbangkan kemungkinan celah itu akan ditutup oleh rintangan yang bergerak, yang seringkali menyebabkan tabrakan atau terjebak.

A* berada di posisi tengah. Tingkat keberhasilannya lebih baik daripada GBFS tetapi jauh di bawah MCTS. Kegagalan A* lebih sering disebabkan oleh timeout (melebihi batas waktu komputasi) daripada tabrakan langsung. Setiap kali jalurnya terhalang, A* mencoba untuk menghitung ulang jalur optimal yang baru. Di lingkungan yang sangat volatil, proses perencanaan ulang ini terjadi begitu sering sehingga waktu komputasi kumulatifnya membengkak, menyebabkan agen gagal mencapai tujuan dalam waktu yang ditentukan.

Analisis ini menggarisbawahi perbedaan filosofi fundamental: pendekatan proaktif MCTS yang mengelola risiko melalui simulasi terbukti jauh lebih unggul dalam menjamin keamanan dan keandalan dibandingkan dengan pendekatan reaktif dari A* dan GBFS yang hanya merespons perubahan setelah terjadi.

C. Analisis Efisiensi: Waktu Komputasi dan Panjang Jalur

Efisiensi algoritma diukur melalui kecepatan komputasi dan optimalitas jalur yang dihasilkan. Tabel berikut menyajikan data kinerja kuantitatif yang dirata-ratakan dari serangkaian *run* pada skenario yang paling menantang (densitas dan volatilitas rintangan tinggi).

Comparison Results

Algorithm	Success Rate	Avg Path Length	Avg Compute Time	Avg Nodes Explored	Avg Replanning
ASTAR	100.0%	1587.0	13.64 ms	5642.4	55.6
GREEDY	100.0%	1743.8	6.18 ms	1800.0	57.2
MCTS	100.0%	786.0	290.32 ms	4572.4	80.4

Gambar 4.3.1. Hasil Kuantitatif Kinerja Algoritma (Sumber: Dokumentasi penulis)

Dari data di atas, beberapa analisis kunci dapat ditarik:

- Monte Carlo Tree Search (MCTS): Secara mengejutkan, MCTS menghasilkan jalur terpendek secara signifikan (786.0), kurang dari setengah panjang jalur yang ditemukan oleh A*. Ini menunjukkan kemampuan luar biasa untuk menemukan rute yang sangat efisien secara fisik di lingkungan yang dinamis. Namun, kualitas jalur superior ini datang dengan biaya komputasi yang sangat tinggi. Dengan waktu komputasi rata-rata 290.32 ms, MCTS adalah algoritma yang paling lambat, lebih dari 45 kali lebih lambat dari Greedy. Selain itu, ia memerlukan jumlah perencanaan ulang tertinggi (80.4), yang mengindikasikan bahwa pendekatan pengambilan keputusan per langkahnya sangat reaktif terhadap perubahan lingkungan.
- Greedy Best-First Search (GBFS): Sesuai dengan sifat "serakah"-nya, GBFS adalah yang tercepat (6.18 ms) dan menjelajahi jumlah node paling sedikit (1800.0). Kecepatan komputasi ini, bagaimanapun, mengorbankan kualitas jalur secara drastis, menghasilkan jalur terpanjang (1743.8) di antara ketiganya. Ini menegaskan profil klasik GBFS sebagai algoritma yang cepat namun tidak optimal.
- A*: Algoritma A* menempati posisi tengah dalam hal panjang jalur (1587.0) dan waktu komputasi (13.64 ms). Sebuah temuan menarik adalah bahwa A* menjelajahi jumlah node terbanyak (5642.4) tetapi memiliki jumlah perencanaan ulang terendah (55.6). Ini menunjukkan bahwa setiap fase perencanaan A* sangat teliti, mengevaluasi banyak kemungkinan untuk menghasilkan jalur awal yang lebih kuat dan lebih jarang gagal dibandingkan dengan jalur yang dihasilkan oleh Greedy. Namun, jalur ini masih jauh dari optimal jika dibandingkan dengan MCTS.

D. Analisis Efisiensi Komputasional (Big-O)

Analisis kompleksitas waktu dan ruang (menggunakan notasi Big-O) memberikan pemahaman teoritis tentang bagaimana kinerja setiap algoritma akan berskala seiring dengan meningkatnya ukuran masalah (misalnya, ukuran grid).

- A*: Dalam kasus terburuk, kompleksitas waktu A* adalah O(b^d), di mana 'b' adalah faktor percabangan (jumlah rata-rata tetangga per node) dan 'd' adalah kedalaman solusi (panjang jalur). Ini menyoroti sensitivitasnya terhadap ukuran ruang pencarian. Kompleksitas ruangnya juga O(b^d) karena harus menyimpan semua node yang dihasilkan dalam memori (open dan closed list), yang bisa menjadi kelemahan signifikan pada peta besar. Kualitas heuristik sangat penting; heuristik yang sempurna mengurangi faktor percabangan efektif menjadi 1, sementara heuristik yang buruk membuatnya tidak lebih baik dari algoritma Dijkstra.
- Greedy Best-First Search (GBFS): Seperti A*, kompleksitas waktu kasus terburuknya adalah O(b^m), di mana 'm' adalah kedalaman maksimum ruang pencarian. Namun, sifat "serakah"-nya seringkali membuatnya menemukan solusi lebih cepat dengan menjelajahi lebih sedikit node, meskipun jalur ini tidak dijamin optimal. Kompleksitas ruangnya umumnya lebih rendah daripada A* karena tidak perlu menyimpan banyak jalur alternatif, hanya berfokus pada yang paling menjanjikan pada setiap langkah.
- Monte Carlo Tree Search (MCTS): Kompleksitas MCTS tidak dinyatakan dalam notasi Big-O tunggal yang terkait dengan ukuran graf, melainkan dalam parameter operasionalnya. Kompleksitasnya sebanding dengan jumlah simulasi (atau iterasi) yang dilakukan untuk setiap keputusan, sebut saja 'n'. Untuk setiap iterasi, ia melakukan seleksi, ekspansi, simulasi, dan propagasi balik. Waktu total dapat diperkirakan sebagai O(n * (T_seleksi + T_ekspansi + T_simulasi + T_propagasi_balik)). Waktu untuk seleksi dan propagasi balik bergantung pada kedalaman pohon yang dibangun (d), sedangkan simulasi bergantung pada panjang rollout. Ini membuat kinerjanya terkait langsung dengan anggaran komputasi yang dialokasikan (waktu per gerakan), sebuah fitur kunci untuk sistem real-time. Namun, kebutuhan memorinya bisa tinggi karena pohon pencarian tumbuh dengan cepat.

E. Komparatif Trade-off dan Implikasi untuk Robot Gudang

Hasil eksperimen ini menyoroti trade-off yang berbeda dan lebih tajam dalam mendefinisikan "optimalitas" untuk sistem dunia nyata. Definisi "optimal" tidak lagi tunggal, melainkan bergantung pada prioritas sistem: efisiensi fisik atau efisiensi komputasi.

Data menunjukkan MCTS adalah juara yang tak terbantahkan dalam optimalitas jalur. Menghasilkan rute yang

dua kali lebih pendek dari A* adalah keuntungan operasional yang sangat besar, yang dapat berarti penghematan energi yang signifikan, pengurangan keausan pada robot, penyelesaian tugas yang lebih cepat, dan lebih sedikit kemacetan di area kerja. Namun, keunggulan ini dibayar dengan kinerja komputasi yang sangat buruk. Sebaliknya, GBFS adalah juara optimalitas komputasi, memberikan keputusan hampir secara instan tetapi dengan jalur yang sangat tidak efisien. A* berada di tengah, tidak unggul secara signifikan di kedua metrik tersebut dalam skenario ini.

Pilihan algoritma untuk robot gudang menjadi pertanyaan strategis tentang alokasi sumber daya.

- Jika sistem gudang memiliki infrastruktur komputasi yang kuat (misalnya, server terpusat yang kuat atau unit pemrosesan canggih di setiap robot) dan prioritas utamanya adalah efisiensi operasional fisik (meminimalkan jarak tempuh robot), maka MCTS adalah pilihan yang sangat menjanjikan. Biaya komputasi yang tinggi dapat dianggap sebagai investasi untuk mendapatkan efisiensi pergerakan yang superior.
- Sebaliknya, jika sistem beroperasi dengan perangkat keras berdaya rendah atau memerlukan latensi keputusan yang sangat rendah (misalnya, di lingkungan yang sangat padat dan cepat berubah di mana penundaan 290 ms tidak dapat diterima), maka kecepatan GBFS mungkin lebih diutamakan, meskipun itu berarti robot akan menempuh rute yang lebih panjang dan berpotensi menyebabkan lebih banyak lalu lintas.

Pilihan antara MCTS dan GBFS mewakili pertukaran mendasar antara investasi komputasi di muka untuk efisiensi fisik jangka panjang versus keputusan latensi rendah dengan biaya operasional fisik yang lebih tinggi.

V. KESIMPULAN DAN SARAN

A. Kesimpulan Perbandingan

Evaluasi kinerja A*, Greedy Best-First Search (GBFS), dan Monte Carlo Tree Search (MCTS) menghasilkan temuan yang jelas dan konsisten mengenai kekuatan dan kelemahan masing-masing:

- Monte Carlo Tree Search (MCTS): Muncul sebagai algoritma yang paling unggul dalam hal kualitas jalur, menghasilkan rute terpendek dengan selisih yang sangat besar. Namun, keunggulan ini membutuhkan biaya komputasi tertinggi, menjadikannya yang paling lambat dari ketiganya.
- Greedy Best-First Search (GBFS): Terbukti sebagai algoritma tercepat dengan upaya komputasi paling sedikit (node yang dieksplorasi). Kecepatan ini, bagaimanapun, menghasilkan jalur terpanjang dan paling tidak efisien.
- A*: Menawarkan keseimbangan antara kualitas jalur dan waktu komputasi, tetapi tidak unggul secara definitif di kedua metrik dibandingkan dengan spesialisasi MCTS dan GBFS. Ia menunjukkan perencanaan awal yang kuat (jumlah perencanaan ulang terendah) tetapi dengan biaya eksplorasi node yang tinggi.

B. Penggunaan dan Rekomendasi

Setiap algoritma memiliki "sweet spot" di mana karakteristiknya paling bersinar. Pemilihan algoritma yang tepat bergantung pada prioritas masalah yang dihadapi.

- A* tetap menjadi pilihan yang solid untuk lingkungan statis di mana jalur optimal absolut diperlukan. Dalam skenario dinamis yang diuji, ia berfungsi sebagai baseline yang seimbang tetapi dikalahkan oleh pendekatan yang lebih terspesialisasi.
- GBFS adalah pilihan yang tepat ketika kecepatan keputusan komputasi adalah satu-satunya prioritas. Ini cocok untuk aplikasi non-kritis di mana latensi sangat rendah lebih penting daripada efisiensi jalur, atau untuk sistem dengan sumber daya komputasi yang sangat terbatas.
- MCTS adalah pilihan superior ketika efisiensi jalur fisik adalah metrik yang paling penting dan ada anggaran komputasi yang memadai. Untuk aplikasi seperti robotika gudang di mana jarak tempuh secara langsung berkaitan dengan konsumsi energi, masa pakai perangkat keras, dan throughput keseluruhan, kemampuan MCTS untuk menemukan jalur yang jauh lebih pendek menjadikannya kandidat yang sangat kuat, asalkan tantangan komputasinya dapat diatasi.

C. Penelitian Selanjutnya

Meskipun MCTS menunjukkan keunggulan yang jelas, masih ada ruang untuk perbaikan dan penelitian lebih lanjut.

Beberapa arah pengembangan yang menjanjikan di masa depan meliputi:

- Hibridisasi Algoritma: Menyelidiki pendekatan hibrida yang menggabungkan kekuatan dari beberapa algoritma. Sebagai contoh, MCTS dapat digunakan untuk perencanaan strategis tingkat tinggi (misalnya, memutuskan koridor atau area umum mana yang akan dilalui untuk menghindari kemacetan), sementara algoritma yang lebih cepat seperti D* Lite atau A* dapat digunakan untuk navigasi taktis tingkat rendah di dalam koridor yang relatif lebih aman tersebut.
- Peningkatan Kinerja MCTS dengan Pembelajaran Mesin: Fase simulasi (rollout) pada MCTS standar seringkali menggunakan kebijakan acak, yang mungkin tidak efisien. Penelitian di masa depan dapat mengeksplorasi penggantian rollout acak ini dengan kebijakan yang dipelajari melalui Reinforcement Learning. Pendekatan seperti Self-Learning MCTS (SL-MCTS), yang menggunakan jaringan saraf untuk memandu pencarian dan mengevaluasi node, dapat secara signifikan meningkatkan efisiensi dan kualitas keputusan MCTS dengan jumlah simulasi yang lebih sedikit.
- Skenario Multi-Agen yang Lebih Kompleks: Penelitian ini berfokus pada agen tunggal yang menavigasi rintangan bergerak. Langkah selanjutnya

adalah memperluas kerangka kerja ini ke skenario Multi-Agent Pathfinding (MAPF) yang sebenarnya, di mana semua entitas yang bergerak adalah agen cerdas dengan tujuan mereka sendiri. Menguji MCTS dalam konteks lifelong MAPF, di mana agen terus-menerus menerima tugas baru dan harus berkoordinasi secara terus-menerus, akan menjadi ujian yang lebih realistis untuk aplikasi gudang.

- Validasi pada Perangkat Keras Fisik: Langkah akhir yang paling penting adalah memvalidasi temuan dari simulasi ini di dunia nyata. Mengimplementasikan algoritma yang paling menjanjikan (MCTS dan mungkin varian hibridanya) pada platform robot fisik akan memberikan bukti definitif tentang keefektifannya dan mengungkap tantangan-tantangan praktis yang tidak muncul dalam simulasi, seperti latensi sensor, ketidakakuratan aktuator, dan dinamika dunia nyata yang lebih kompleks.

LAMPIRAN

Kode sumber lengkap dalam makalah ini dapat diakses melalui tautan repository GitHub berikut: https://github.com/andi-frame/stima-moving-maze.

UCAPAN TERIMA KASIH

Penulis ingin menyampaikan rasa syukur kepada Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, yang memungkinkan penulis untuk menyelesaikan makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada Menterico Adrian, S.T, M.T, Dr. Nur Ulfa Maulidevi, dan Dr. Ir. Rinaldi Munir, M.T., sebagai dosen mata kuliah IF2211 Strategi Algoritma yang telah memberikan bimbingan, saran, serta masukan yang sangat berharga selama proses pembelajaran dan penulisan makalah ini. Dukungan dan pengetahuan yang diberikan sangat membantu penulis dalam menyelesaikan tugas ini. Penulis juga ingin mengucapkan terima kasih yang mendalam kepada orang tua dan keluarga yang senantiasa memberikan dukungan moral dan materiil serta doa tanpa henti. Tidak lupa, penulis juga berterima kasih kepada teman-teman dan rekan-rekan seperjuangan yang selalu memberikan semangat dan bantuan, baik secara langsung maupun tidak langsung dalam proses penyusunan makalah ini. Penulis menyadari bahwa makalah ini belum sempurna, oleh karena itu kritik dan saran yang konstruktif sangat diharapkan untuk perbaikan di masa depan. Terima kasih.

REFERENSI

[1] Supply Chain Today. (n.d.). Autonomous robots revolutionizing supply chain. Diakses pada 24 Juni 2025, dari https://www.supplychaintoday.com/autonomous-robots-revolutionizing-supply-chain/

[2] Zewe, A. (2024, 27 Februari). New AI model could streamline operations in a robotic warehouse. MIT News. Diakses pada 24 Juni 2025, dari https://news.mit.edu/2024/new-ai-model-could-streamline-operations-robotic-warehouse-0227

[3] Anonymous. (n.d.). AD algorithm, a path finding algorithm that works in dynamic environment for warehouse robot*. CORE. Diakses pada 24 Juni 2025, dari https://core.ac.uk/download/pdf/322377168.pdf

[4] PYMNTS. (2024, 28 Februari). MIT AI model promises to simplify path planning in warehouses. The Robot Report. Diakses pada 24 Juni 2025, dari https://www.therobotreport.com/mit-ai-model-promises-to-simplify-path-planning-in-warehouses/

[5] Satria, H., Primanita, A., & Syahroyni, M. (2018). Analisa perbandingan algoritma A* dan dynamic pathfinding algorithm dengan dynamic pathfinding algorithm untuk NPC pada car racing game. Jurnal Teknologi Informasi dan Ilmu Komputer, 5(1), 95-103. https://doi.org/10.25126/jtiik.201851544

[6] van den Berg, J., Lin, M. C., & Manocha, D. (n.d.). Real-time path planning for multiple virtual agents in complex dynamic scenes. University of North Carolina at Chapel Hill. Diakses pada 24 Juni 2025, dari http://gamma.cs.unc.edu/crowd/paper300.pdf

[7] Skrynnik, A., Andreychuk, A., & Yakovlev, K. (2023). A decentralized multi-agent pathfinding with Monte Carlo tree search. arXiv preprint arXiv:2312.15908. https://doi.org/10.48550/arXiv.2312.15908

[8] Anonymous. (2024). A comparative analysis of A and D* Lite in dynamic environments*. KTH Royal Institute of Technology.

[9] Bency, V. J., & K, S. (2025). Monte Carlo Tree Search with Velocity Obstacles for safe and efficient motion planning in dynamic environments. arXiv preprint arXiv:2501.09649.

[10] Chen, Y.-J., Jhong, B.-G., & Chen, M.-Y. (2023). A real-time path planning algorithm based on the Markov decision process in a dynamic environment for wheeled mobile robots. Drones, 7(4), 249. https://doi.org/10.3390/drones7040249

[11] Anonymous. (2022). A comparisonalgorithm for code optimization. The SAI Organization.

[12] Green, M. C., et al. (2018). Search-based procedural content generation. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 14(1), 1-7.

[13] Pas, A., & Spaan, M. T. J. (2014). A comparison of Monte Carlo tree search and mathematical optimization for large scale dynamic resource allocation. arXiv preprint arXiv:1405.5498.

[14] Anonymous. (2023). Evaluation of popular path planning algorithms. International Journal of Electronics and Telecommunications, 69(4), 1-8. https://doi.org/10.24425/ijet.2023.147699

[15] Patel, A. (n.d.). A Pathfinding*. Red Blob Games. Diakses pada 24 Juni 2025, dari (http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html)

[16] Patel, A. (n.d.). Introduction to A*. Red Blob Games. Diakses pada 24 Juni 2025, dari https://www.redblobgames.com/pathfinding/a-star/introduction.html

[17] Simplilearn. (2023, 19 Desember). A search algorithm*. Simplilearn. Diakses pada 24 Juni 2025, dari https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

[18] Algocademy. (n.d.). An introduction to a pathfinding algorithm. Algocademy. Diakses pada 24 Juni 2025, dari https://algocademy.com/blog/an-introduction-to-a-pathfinding-algorithm/

[19] GeeksforGeeks. (2023, 21 November). A algorithm and its heuristic search strategy in Artificial Intelligence*. GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/artificial-intelligence/a-algorithm-and-its-heuristic-search-strategy-in-artificial-intelligence/

[20] Anonymous. (n.d.). A Pathfinding*. Pace University. Diakses pada 24 Juni 2025, dari https://csis.pace.edu/~benjamin/teaching/cs627/webfiles/Astar.pdf

[21] Computer Science Bytes. (n.d.). The A pathfinding algorithm*. Computer Science Bytes. Diakses pada 24 Juni 2025, dari http://www.computersciencebytes.com/array-variables/graphs/the-a-pathfinding-algorithm/

[22] AI Smart Class. (n.d.). What is A algorithm?*. AI Smart Class. Diakses pada 24 Juni 2025, dari https://aismartclass.net/blog/what-is-a-star-algorithm/

[23] GeeksforGeeks. (2023, 25 Oktober). Difference between best-first search and A search?*. GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/difference-between-best-first-search-and-a-search/

[24] Codecademy. (n.d.). Greedy best-first search. Codecademy Docs. Diakses pada 24 Juni 2025, dari https://www.codecademy.com/resources/docs/ai/search-algorithms/greedy-best-first-search

[25] GeeksforGeeks. (2024, 20 Mei). Greedy best-first search in AI. GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/artificial-intelligence/greedy-best-first-search-in-ai/

[26] GeeksforGeeks. (2023, 1 Agustus). Greedy best-first search. GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/

[27] Anonymous. (2021). Comparison of A* algorithm and Greedy Best Search in searching fifteen puzzle solution. ResearchGate.

[28] Multiple authors. (2018). What are the differences between A and greedy best-first search?* [Unggahan forum online]. AI Stack Exchange. Diakses pada 24 Juni 2025, dari https://ai.stackexchange.com/questions/8902/what-are-the-differences-between-a-and-greedy-best-first-search

[29] Multiple authors. (2015). A efficiency vs Greedy Best First*. [Unggahan forum online]. Stack Overflow. Diakses pada 24 Juni 2025, dari https://stackoverflow.com/questions/28655197/a-efficiency-vs-greedy-best-first

[30] Sharma, A. (2024, 10 Juni). Understanding the Greedy Best-First Search (GBFS) algorithm in Python. Analytics Vidhya. Diakses pada 24 Juni 2025, dari https://www.analyticsvidhya.com/blog/2024/06/understanding-the-greedy-best-first-search-gbfs-algorithm-in-python/

[31] GeeksforGeeks. (2023, 1 Agustus). Greedy best-first search. GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/greedy-best-first-search-algorithm/

[32] Chen, C., et al. (2023). A self-learning Monte Carlo tree search algorithm for robot path planning. Frontiers in Neurorobotics, 17. https://doi.org/10.3389/fnbot.2023.1039644

[33] Radke, P. (2022, 2 Agustus). What is Monte Carlo tree search?. Built In. Diakses pada 24 Juni 2025, dari https://builtin.com/machine-learning/monte-carlo-tree-search

[34] GeeksforGeeks. (2023, 2 Agustus). ML | Monte Carlo Tree Search (MCTS). GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/machine-learning/ml-monte-carlo-tree-search-mcts/

[35] Browne, C. B., et al. (2012). A survey of Monte Carlo Tree Search methods. IEEE Transactions on Computational Intelligence and AI in Games, 4(1), 1-43.

[36] Multiple authors. (2019). Pathfinding with non-deterministic transitions between nodes. [Unggahan forum online]. Gamedev Stack Exchange. Diakses pada 24 Juni 2025, dari https://gamedev.stackexchange.com/questions/166976/pathfinding-with-non-deterministic-transitions-between-nodes

[37] GeeksforGeeks. (2023, 2 Agustus). ML | Monte Carlo Tree Search (MCTS). GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/machine-learning/ml-monte-carlo-tree-search-mcts/

[38] Baeldung. (n.d.). The A algorithm*. Baeldung on Computer Science. Diakses pada 24 Juni 2025, dari https://www.baeldung.com/cs/a-star-algorithm

[39] Number Analytics. (n.d.). A search: The ultimate pathfinding solution*. Number Analytics. Diakses pada 24 Juni 2025, dari https://www.numberanalytics.com/blog/a-star-search-ultimate-pathfinding-solution

[40] GeeksforGeeks. (2023, 25 Oktober). Difference between best-first search and A search?*. GeeksforGeeks. Diakses pada 24 Juni 2025, dari https://www.geeksforgeeks.org/difference-between-best-first-search-and-a-search/

[41] Multiple authors. (2016). What's the time complexity of Monte Carlo Tree Search? [Unggahan forum online]. Computer Science Stack Exchange. Diakses pada 24 Juni 2025, dari https://cs.stackexchange.com/questions/51726/whats-the-time-complexity-of-monte-carlo-tree-search

[42] Multiple authors. (2018). What are the differences between A and greedy best-first search?* [Unggahan forum online]. AI Stack Exchange. Diakses pada 24 Juni 2025, dari https://ai.stackexchange.com/questions/8902/what-are-the-differences-between-a-and-greedy-best-first-search

[43] Chen, C., et al. (2023). A self-learning Monte Carlo tree search algorithm for robot path planning. Frontiers in Neurorobotics, 17. https://doi.org/10.3389/fnbot.2023.1039644

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jatinangor, 24 Juni 2025

Andi Farhan Hidayat
13523128