Optimalisasi Pencarian Jalur dalam Labirin Menggunakan Algoritma A*

13522143-Muhammad Fatihul Irhab
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522143@std.stei.itb.ac.id

***Abstract*— Algoritma A* adalah salah satu metode terbaik dan paling efisien untuk mencari jalan, terutama saat kita harus keluar dari labirin yang rumit. Makalah ini membahas cara kerja dan optimasi algoritma A* dalam menemukan jalan di berbagai jenis labirin. Dengan menggunakan yang didasarkan pada jarak Manhattan, algoritma A* bisa memberikan solusi yang cepat dan tepat. Pendekatan ini memungkinkan A* untuk menghitung jarak terpendek dari titik awal ke tujuan dengan mempertimbangkan jarak yang sudah ditempuh dan perkiraan jarak yang masih harus ditempuh. Dalam eksperimen yang dilakukan, algoritma ini diuji pada berbagai labirin dengan tingkat kesulitan yang berbeda. Tujuan pengujian ini adalah untuk melihat seberapa baik algoritma A* bekerja dalam hal kecepatan dan akurasi dalam menemukan jalan terpendek. Hasil pengujian menunjukkan bahwa algoritma A* tidak hanya cepat dalam menemukan jalan terpendek, tetapi juga sangat fleksibel ketika bentuk labirin berubah. Selain itu, kemampuan adaptif algoritma A* memungkinkan penyesuaian cepat terhadap kondisi yang berubah, menjadikannya alat yang sangat efektif untuk mencari jalan dalam situasi yang dinamis.**

***Keywords*—Algoritma A*, Labirin, Rute Pencarian, Jarak Manhattan**

I. PENDAHULUAN

Labirin adalah salah satu contoh permasalahan yang kompleks dalam rute pencarian jalur. Labirin bisa sangat bervariasi, mulai dari yang sederhana hingga yang sangat rumit dengan banyak rintangan dan jalan buntu. Mencari jalur dalam labirin memerlukan algoritma yang tidak hanya bisa menemukan jalan terpendek, tetapi juga mampu beradaptasi dengan perubahan struktur labirin. Ini membuat labirin menjadi contoh permasalahan yang bagus untuk menguji kemampuan algoritma dalam mencari rute pencarian jalur.

Gambar 1.1 Sebuah Labirin

Sumber: https://depositphotos.com/id/vector/maze-labyrinth-game-193034776.html

Pencarian jalur adalah masalah yang sering kita temukan dalam kehidupan sehari-hari, masalah ini ada dalam banyak bidang, seperti game, robotika, hingga navigasi kendaraan. Dalam konteks labirin, misalnya, karakter atau robot perlu menemukan jalan terpendek untuk sampai ke tujuan, sambil menghindari rintangan dan menghadapi berbagai tantangan. Begitu juga dengan robot yang harus bergerak melalui labirin, robot perlu menemukan jalur yang efisien tanpa menabrak dinding atau hambatan lainnya. Karena itu, kita memerlukan algoritma pencarian jalur yang cepat, efisien, dan fleksibel.

Algoritma A* adalah salah satu algoritma pencarian jalur yang paling populer dan efisien. Algoritma ini menggabungkan kekuatan dari algoritma UCS dan Greedy Best-First-Search, menggunakan fungsi heuristik untuk memprediksi biaya terendah dari titik awal ke tujuan. Ini memungkinkan pencarian jalur yang cepat dan optimal.

Pada makalah ini akan dibahas cara mengoptimalkan algoritma A* dalam menemukan jalur terpendek di berbagai jenis labirin. Selain itu, penulis akan mengukur kinerja algoritma A* menggunakan jarak Manhattan sebagai fungsi heuristik. Dengan makalah ini, penulis berharap bisa memberikan kontribusi dalam bidang pencarian jalur, menawarkan solusi yang efisien untuk pencarian jalur di labirin yang kompleks, memberikan wawasan tentang bagaimana mengoptimalkan algoritma A* dalam situasi labirin yang dinamis, serta menyediakan data hasil tentang kinerja algoritma A* dengan menggunakan jarak Manhattan.

II. TEORI DASAR

A. Pencarian Jalur

Pencarian jalur adalah proses menemukan rute terbaik dari satu titik ke titik lain dalam suatu area, biasanya berdasarkan kriteria rute paling pendek atau cepat. Aplikasi pencarian jalur mencakup game, navigasi robot, dan sistem GPS. Algoritma pencarian jalur harus adaptif dan fleksibel untuk menangani rintangan dan situasi yang berubah-ubah. Beberapa algoritma terkenal adalah UCS, Greedy Best-First-Searh, A*, Breadth-First Search (BFS), dan Depth-First Search (DFS). Efisiensi waktu dan sumber daya sangat penting, terutama dalam aplikasi real-time seperti game dan navigasi robot. Penelitian terus dilakukan untuk menciptakan algoritma yang lebih efisien dan adaptif, memastikan pencarian jalur tetap menjadi alat yang andal dalam berbagai aplikasi.

B. *Algoritma Pencarian Jalur*

Ada beberapa algoritma yang digunakan untuk pencarian jalur, di antaranya adalah:

1. Algoritma UCS (Uniform Cost Search)

Algoritma ini memilih jalur dengan biaya terendah pada setiap langkahnya, tanpa terlalu memperhatikan seberapa jauh suatu tempat dari titik awal. UCS fokus pada biaya dari titik awal ke setiap simpul secara individual, sehingga bisa menemukan jalur yang lebih efisien dalam beberapa situasi, terutama ketika ada jalur yang biayanya jauh lebih murah ke tujuan daripada jalur lainnya. Meskipun demikian, UCS masih membutuhkan waktu yang signifikan untuk mengeksplorasi seluruh graf jika grafnya sangat besar. Implementasi untuk algoritma ini menggunakan f(n) = g(n) dimana g(n) adalah biaya untuk setiap simpulnya yang dihitung dari titik awal sampai titik n.

2. Algoritma Greedy Best-First-Search

Algoritma Greedy Best-First-Search memilih langkah selanjutnya berdasarkan nilai heuristik lokal, fokus pada jalur yang dianggap paling menguntungkan saat itu tanpa memperhitungkan secara keseluruhan apakah itu akan menghasilkan jalur terpendek. Meskipun pendekatannya yang cepat dan sederhana cocok untuk pencarian jalur dalam permasalahan yang besar, algoritma ini tidak memberikan jaminan dalam menemukan jalur terpendek. Implementasi untuk algoritma ini menggunakan f(n) = h(n) dimana h(n) adalah perkiraan biaya untuk setiap simpulnya yang dihitung dari titik n sampai titik tujuan.

3. Algoritma A*

Algoritma A* adalah algoritma pencarian yang digunakan untuk menemukan jalur terpendek dari titik awal ke tujuan dalam graf. Algoritma ini menggabungkan keuntungan dari algoritma UCS dan algoritma Greedy Best-First-Search dengan memanfaatkan fungsi heuristik yang memperkirakan jarak tersisa ke tujuan. A* bekerja dengan mengevaluasi setiap simpul berdasarkan biaya kumulatif dari titik awal dan perkiraan biaya ke tujuan, memastikan jalur yang ditemukan tidak hanya efisien tetapi juga optimal. Heuristik yang digunakan harus memenuhi kondisi tertentu agar A* tetap optimal dan lengkap, yaitu tidak melebih-lebihkan jarak sebenarnya ke tujuan. Algoritma ini menggunakan dua fungsi biaya yaitu g(n) sebagai biaya dari titik awal ke titik n dan h(n) sebagai perkiraan biaya dari titik n ke titik tujuan, sehingga total biaya yang dihitung adalah f(n) = g(n) + h(n).

4. Algoritma BFS (Breadth-First Search)

Algoritma BFS adalah algoritma pencarian jalur yang mengeksplorasi semua simpul pada tingkat tertentu sebelum beralih ke tingkat berikutnya. BFS menggunakan struktur data antrian (queue) untuk melacak simpul-simpul yang akan dieksplorasi berikutnya. Algoritma ini sangat efektif untuk menemukan jalur terpendek dalam graf yang tidak berbobot (unweighted graph), karena ia menjelajahi simpul-simpul secara bertahap dari titik awal, memastikan bahwa simpul-simpul terdekat dieksplorasi terlebih dahulu. BFS dijamin menemukan jalur terpendek dalam graf yang tidak berbobot, tetapi bisa menjadi tidak efisien dalam graf yang sangat besar atau graf dengan banyak tingkat.

5. Algoritma DFS (Depth-First Search)

Algoritma DFS adalah algoritma pencarian jalur yang mengeksplorasi satu jalur sedalam mungkin sebelum kembali dan mencoba jalur lain. DFS menggunakan struktur data tumpukan (stack) untuk melacak simpul-simpul yang sedang dieksplorasi. DFS sangat berguna dalam aplikasi seperti menemukan semua jalur dari titik awal ke titik tujuan, mendeteksi siklus dalam graf, atau memecahkan masalah yang memerlukan penelusuran mendalam. Namun, DFS tidak menjamin menemukan jalur terpendek karena ia dapat terus mengeksplorasi jalur yang panjang tanpa memeriksa jalur yang lebih pendek terlebih dahulu.

C. *Jarak Manhattan*

Jarak Manhattan adalah cara untuk mengukur jarak antara dua titik dalam ruang dua dimensi. Namanya diambil dari jaringan jalan di Kota New York, di mana jarak antara dua titik dihitung sebagai jumlah total perpindahan horizontal dan vertikal yang diperlukan untuk mencapai tujuan, tanpa memperhitungkan pergerakan diagonal. Secara matematis, Jarak Manhattan antara dua titik (x1, y1) dan (x2, y2) dalam ruang koordinat dua dimensi didefinisikan sebagai |x1 - x2| + |y1 - y2|. Metrik ini sering digunakan dalam berbagai bidang, seperti komputasi graf, perencanaan rute, dan pemrosesan gambar, karena sifatnya yang

D. *Labirin*

Labirin adalah struktur atau graf yang kompleks dengan banyak jalur dan rintangan. Labirin dapat memiliki berbagai bentuk dan ukuran, dari yang sederhana dengan sedikit rintangan hingga yang sangat kompleks dengan banyak jalan buntu dan jalur melingkar. Dalam konteks penelitian ini, labirin digunakan sebagai model untuk menguji kemampuan algoritma pencarian jalur. Jenis-jenis labirin:

1. Labirin Klasik

Labirin klasik adalah tipe labirin yang paling umum dan tradisional. Labirin ini biasanya memiliki satu titik masuk dan satu titik keluar, dengan satu jalur yang benar yang harus ditemukan di antara banyak jalan buntu. Labirin klasik sering digunakan dalam permainan papan dan puzzle, serta dalam berbagai aplikasi hiburan. Kesulitan dalam labirin klasik terletak pada kemampuan untuk mengidentifikasi jalur yang benar di antara banyak pilihan yang salah, yang memerlukan pemikiran logis dan keterampilan navigasi yang baik.

2. Labirin Grid

Labirin grid dibangun di atas grid persegi, di mana setiap sel dalam grid bisa menjadi jalan yang bisa dilewati atau rintangan yang tidak bisa

dilewati. Labirin jenis ini sering digunakan dalam penelitian algoritma pencarian jalur karena kesederhanaan strukturnya dan kemudahan implementasi. Pada labirin grid, setiap langkah hanya bisa dilakukan dalam arah horizontal atau vertikal, membuatnya ideal untuk menguji algoritma seperti A* yang menggunakan fungsi heuristik seperti jarak Manhattan. Labirin grid juga sering muncul dalam konteks robotika dan game video, di mana robot atau karakter harus menemukan jalan melalui area yang dibatasi oleh rintangan.

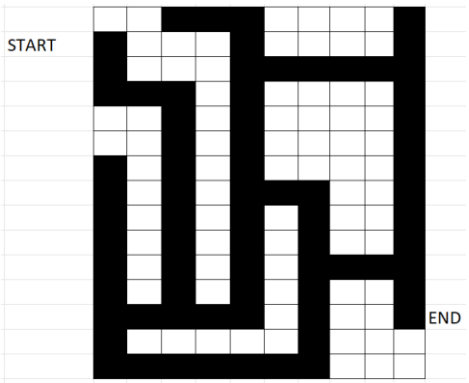
3. Labirin Dinamis

   Labirin dinamis adalah jenis labirin yang bentuk dan rintangannya dapat berubah seiring waktu. Perubahan ini bisa terjadi secara acak atau mengikuti pola tertentu, menambahkan tantangan tambahan bagi algoritma pencarian jalur. Labirin dinamis sangat relevan dalam aplikasi dunia nyata, seperti navigasi robot di lingkungan yang berubah-ubah atau dalam permainan video di mana tingkat kesulitan meningkat seiring berjalannya waktu. Untuk mengatasi tantangan dalam labirin dinamis, algoritma pencarian jalur harus memiliki kemampuan adaptasi yang tinggi dan dapat merespons perubahan dengan cepat dan efisien.

III. PEMBAHASAN DAN IMPLEMENTASI

Dalam pembahasan ini, penulis akan menggunakan satu labirin untuk menguji apakah algoritma A* dapat menemukan jalur terpendek secara optimal. Implementasi algoritma A* akan dilakukan pada labirin tersebut, dan hasilnya akan dianalisis untuk menilai efektivitas dan efisiensi algoritma dalam menyelesaikan masalah pencarian jalur terpendek.

A. Representasi Labirin



Gambar 3.1 Labirin Grid

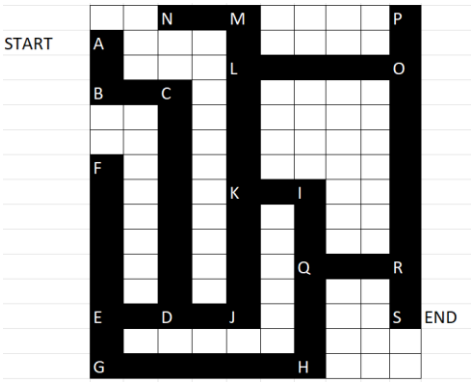
Sumber: Dokumen pribadi

Dalam gambar 3.1, labirin di representasikan ke dalam sebuah grid, hal ini perlu dilakukan karena dalam implementasi algoritma A* pada permasalahan labirin, kita perlu memetakan labirin ke dalam bentuk representasi grid untuk mempermudah pencarian rute jalur optimal, berikut adalah keuntungan representasi labirin ke dalam bentuk grid:

1. Kemudahan dalam melakukan implementasi, representasi grid membuat struktur dari labirin menjadi lebih mudah untuk dipahami dan dimplementasikan dalam kode, hal ini diperlukan karena setiap sel dalam grid bisa dengan mudah diakses dan diperiksa statusnya (apakah dapat dilalui atau tidak). Dengan representasi grid memungkinkan pengkodean yang lebih sederhana dan lebih terstruktur, di mana setiap sel grid dapat direpresentasikan sebagai node dengan koordinat tertentu (x, y).
2. Efisiensi dalam perhitungan, dengan menggunakan grid, menghitung biaya pergerakan dari satu titik ke titik lainnya menjadi lebih sederhana, hal ini karena pergerakan horizontal dan vertikal memiliki biaya tetap, dan jarak antara dua titik bisa dihitung dengan mudah menggunakan jarak Manhattan. Representasi grid memastikan bahwa setiap langkah dari satu sel ke sel tetangga hanya memerlukan satu satuan biaya, sehingga perhitungan g(n) dan h(n) menjadi lebih efisien.
3. Fleksibilitas, representasi grid dengan mudah disesuaikan untuk berbagai ukuran dan bentuk labirin. Selain itu, perubahan atau pembaruan dalam struktur labirin, seperti penambahan atau penghapusan rintangan, dapat dilakukan dengan mudah. Cukup dengan menandai sel-sel tertentu dalam grid sebagai tidak dapat dilalui, kita dapat mengubah dinamika pencarian rute tanpa memerlukan perubahan besar dalam algoritma.

Dengan representasi grid, algoritma A* dapat dijalankan dengan efisien, memastikan jalur yang ditemukan tidak hanya terpendek tetapi juga optimal. Representasi ini memberikan dasar yang kuat bagi algoritma untuk melakukan pencarian jalur yang efektif dan efisien dalam berbagai jenis labirin dan kondisi.

B. Representasi Node



Gambar 3.2 Representasi node dalam labirin

Sumber: Dokumen pribadi

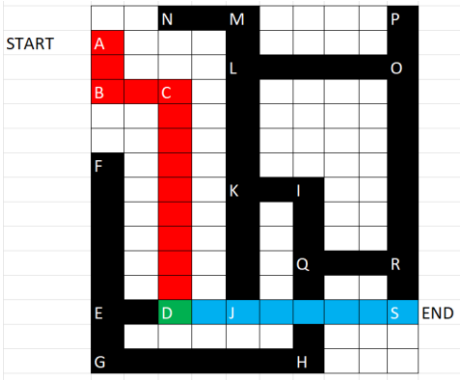
Dalam gambar 3.2, setiap huruf melambangkan node yang ada dalam labirin ini, node dibentuk setiap persimpangan dan jalur buntu. Representasi node ini sangat penting untuk implementasi algoritma A*, karena algoritma ini membutuhkan pemetaan yang jelas dari setiap titik kritis dalam labirin untuk menentukan jalur terpendek dengan efisien. Berikut adalah alasan pentingnya rpresentasi node dalam pemecahan labirin ini:

1. Mengidentifikasi titik kritis, Setiap persimpangan dan jalan buntu dalam labirin dianggap sebagai node. Ini adalah titik-titik di mana keputusan harus dibuat tentang arah yang harus diambil selanjutnya. Dengan mengidentifikasi titik-titik ini sebagai node, kita dapat memecah labirin menjadi bagian-bagian yang lebih kecil dan lebih mudah dikelola. Hal ini sangat membantu dalam mengoptimalkan pencarian jalur karena kita hanya perlu fokus pada titik-titik kritis daripada setiap sel dalam labirin.
2. Menyederhanakan dalam perhitungan, dengan memetakan labirin menjadi node-node, perhitungan biaya perjalanan dari satu titik ke titik lainnya menjadi lebih sederhana. Algoritma A* dapat dengan mudah menghitung biaya g(n) dari titik awal ke node saat ini dan biaya heuristik h(n) dari node saat ini ke tujuan, hal ini terjadi karena setiap langkah dari satu node ke node yang berdekatan memiliki biaya tetap.
3. Memnfasilitasi pencarian jalur, node-node ini membentuk graf yang dapat dieksplorasi oleh algoritma A*. Algoritma A* bekerja dengan mengevaluasi setiap node berdasarkan biaya kumulatif g(n) dan perkiraan biaya h(n), sehingga memastikan jalur yang ditemukan tidak hanya efisien tetapi juga optimal. Representasi ini memungkinkan algoritma untuk memanfaatkan struktur data seperti priority queue untuk menyimpan dan mengakses node-node yang perlu dievaluasi dengan efisien.

Langkah membentuk node-node yang ada dalam labirin:

1. Mengidentifikasi persimpangan, persimpangan adalah titik di mana ada satu atau lebih arah yang dapat dipilih. Setiap persimpangan perlu dijadikan sebuah node karena di titik-titik ini algoritma perlu membuat keputusan tentang arah yang harus diambil selanjutnya.
2. Mengidentifikasi jalur buntu, jalur buntu memberikan informasi bahwa jalur ini sudah tidak dapat digunakan untuk mencapai tujuan, hal ini sangat membantu algoritma untuk melakukan pencarian ke node selanjutnya, sehingga setiap jalur buntu harus dijadikan ke dalam sebuah node.

C. Perhitungan Biaya dan Nilai Heuristik



Gambar 3.3 Perhitungan dalam labirin

Sumber: Dokumen pribadi

Dalam gambar 3.3, kita menggunakan dua jenis nilai untuk membantu algoritma A* menemukan jalur terpendek: biaya g(n) dan nilai heuristik h(n). Warna merah pada gambar menunjukkan total biaya g(n) yang dibutuhkan untuk bergerak dari titik awal A hingga titik D, sedangkan warna biru menunjukkan nilai heuristik h(n) dari titik D terhadap titik akhir S.

Biaya g(n) adalah biaya kumulatif yang dibutuhkan untuk mencapai node n dari titik awal. Biaya ini biasanya dihitung sebagai jumlah langkah horizontal dan vertikal yang diperlukan untuk mencapai node tersebut. Dalam konteks grid, setiap langkah horizontal atau vertikal memiliki biaya tetap yaitu satu. Contoh menghitung biaya g(n) dari titik A ke titik D yaitu, dibutuhkan 11 (langkah ke bawah) + 2 (langkah ke kanan) = 13 langkah dibutuhkan. Berikut adalah biaya g(n) antara dua buah titik:

Titik Awal	Titik Akhir	Biaya g(n)
A	B	2
B	C	2
C	D	9
D	E	2
D	J	2
E	F	6
E	G	2
G	H	6
H	Q	4
J	K	5
K	I	2
K	L	5
I	Q	3
L	M	2
L	O	5
M	N	2
O	P	2
O	R	8
R	S	2

Nilai heuristik h(n) adalah perkiraan biaya yang diperlukan untuk mencapai tujuan dari node n. Untuk algoritma A*, nilai heuristik ini biasanya dihitung menggunakan jarak Manhattan, yang hanya mempertimbangkan perpindahan horizontal dan vertikal. Contoh menghitung biaya h(n) dari titik D yaitu, dengan menggunakan jarak Manhattan |x_end - x| + |y_end - y| = |9 - 2| + |12 - 12| = 7 langkah yang dibutuhkan. Berikut adalah nilai heuristik h(n) dari setiap titik terhadap titik akhir:

Titik	Perhitungan Manhattan	Nilai h(n)
A	|9 - 0| + |12 – 1|	20
B	|9 - 0| + |12 – 3|	18
C	|9 - 2| + |12 – 7|	16
D	|9 - 2| + |12 – 12|	7
E	|9 - 0| + |12 – 12|	9
F	|9 - 0| + |12 – 6|	15
G	|9 - 0| + |12 – 14|	11
H	|9 - 6| + |12 – 14|	5
I	|9 - 6| + |12 – 7|	8
J	|9 - 4| + |12 – 12|	6
K	|9 - 4| + |12 – 7|	10
L	|9 - 4| + |12 – 2|	15
M	|9 - 4| + |12 – 0|	17
N	|9 - 2| + |12 – 0|	19
O	|9 - 9| + |12 – 2|	10
P	|9 - 9| + |12 – 0|	12
Q	|9 - 6| + |12 – 10|	5
R	|9 - 9| + |12 – 10|	2
S	|9 - 9| + |12 – 12|	0

*D. Implementasi Algoritma A**

Implementasi algoritma A* untuk menemukan jalur terpendek pada labirin yang telah direpresentasikan dalam bentuk grid dan node-node kritis dilakukan dengan melalui beberapa iterasi. Setiap iterasi melibatkan pemilihan simpul dengan nilai f(n) terkecil dari daftar simpul hidup (open list) dan ekspansi simpul tersebut. Implementasi algoritma A* melibatkan beberapa komponen utama:

1. Struktur node: Setiap node dalam grid diwakili oleh struktur data yang mencakup posisi, biaya g(n), nilai heuristik h(n), dan total biaya f(n). Node juga memiliki referensi ke node induknya untuk rekonstruksi jalur. Struktur data ini memungkinkan kita untuk melacak jalur yang telah dilalui dan biaya yang terkait dengan masing-masing node.
    - Posisi: Koordinat (x, y) dalam grid.
    - Biaya g(n): Total biaya dari titik awal hingga node ini.
    - Nilai heuristik h(n): Perkiraan biaya dari node ini ke tujuan.
    - Total biaya f(n): Jumlah dari g(n) dan h(n), yaitu f(n) = g(n) + h(n).
    - Node induk: Referensi ke node sebelumnya untuk rekonstruksi jalur.
2. Open List: Open list menyimpan node-node yang akan dievaluasi. Open list diimplementasikan sebagai heap (priority queue) untuk memungkinkan pemilihan node dengan nilai f(n) terkecil secara efisien. Dengan menggunakan priority queue, kita dapat memastikan bahwa node dengan nilai f(n) terkecil selalu dievaluasi terlebih dahulu, yang penting untuk menemukan jalur terpendek dengan cepat.
3. Fungsi Heuristik: Fungsi heuristik h(n) digunakan untuk memperkirakan biaya dari node saat ini ke tujuan. Dalam kasus ini, kita menggunakan jarak Manhattan karena labirin hanya memungkinkan pergerakan horizontal dan vertikal. Fungsi heuristik ini haruslah admissible, artinya tidak boleh melebihi biaya aktual untuk mencapai tujuan, agar A* tetap optimal.
4. Fungsi Biaya: Fungsi biaya g(n) digunakan untuk mendapatkan total biaya dari titik awal hingga titik n. Dalam kasus ini biaya satu grid dinilai sebagai satu biaya.
5. Fungsi Evaluasi: Fungsi evaluasi f(n) digunakan untuk nilai dari setiap node, yang nantinya akan diurutkan dalam priority queue. Fungsi evaluasi f(n) adalah kombinasi dari dua komponen yaitu g(n) dan h(n), sehinggan fungsi evaluasi f(n) dirumuskan sebagai f(n) = g(n) + h(n).
6. Algoritma pencarian rute: Algoritma A* bekerja dengan meng-ekspansi node dari open list, memperbarui biaya g(n) dan h(n) untuk setiap tetangga, dan memindahkan node ke closed list setelah dievaluasi. Jika node tujuan ditemukan, jalur terpendek direkonstruksi dengan melacak kembali dari node tujuan ke node awal.

Berikut adalah langkah-langkah rinci implementasi algoritma A*:

Iterasi	Simpul Ekspan	Simpul Hidup (Priority Queue)
1	A	B(20)
2	B	C(20)
3	C	D(20)
4	D	J(21), E(24)
5	J	E(24), K(30)
6	E	G(28), K(30), F(36)
7	G	H(28), K(30), F(36)
8	H	K(30), Q(32), F(36)
9	K	I(30), Q(32), F(36), L(40)
10	I	Q(30), F(36), L(40)
11	Q	R(30), F(36), L(40)
12	R	S(30), F(36), L(40)
13	S	Goal

Maka dari langkah-langkah di atas didapat jalur terpendek, berdasarkan algoritma A* yaitu A->B->C->D->J->K->I->Q->R->S

IV. KESIMPULAN

Berdasarkan pembahasan dan implementasi algoritma A* untuk menemukan jalur terpendek pada labirin yang telah direpresentasikan dalam bentuk grid dan node-node kritis, dapat diambil beberapa kesimpulan sebagai berikut:

1. Efektivitas Algoritma A*, algoritma A* terbukti efektif dalam menemukan jalur terpendek pada labirin yang kompleks. Dengan memanfaatkan representasi grid, algoritma dapat mengeksplorasi dan memetakan labirin dengan lebih mudah, sehingga mampu menemukan jalur optimal dari titik awal ke titik tujuan.

2. Efisiensi Algoritma A*, implementasi algoritma A* menunjukkan efisiensi yang baik dalam perhitungan biaya dan nilai heuristik. Penggunaan jarak Manhattan sebagai fungsi heuristik membantu memperkirakan biaya dengan akurat, sementara fungsi biaya g(n) memastikan bahwa setiap langkah pergerakan dihitung dengan benar. Struktur open list yang diimplementasikan sebagai heap (priority queue) memungkinkan pemilihan node dengan nilai f(n) terkecil secara efisien, mempercepat proses pencarian jalur.

3. Pentingnya Representasi Grid dan Node, representasi labirin dalam bentuk grid dan node-node kritis memberikan dasar yang kuat bagi algoritma A* untuk melakukan pencarian jalur. Representasi grid mempermudah implementasi dan perhitungan biaya pergerakan, sementara representasi node membantu mengidentifikasi titik-titik kritis dalam labirin, sehingga algoritma dapat mengevaluasi setiap node dengan lebih terstruktur.

4. Iterasi dan Ekspansi Node, melalui beberapa iterasi, algoritma A* mampu mengevaluasi dan mengekspansi node-node yang ada dalam labirin. Proses iterasi yang melibatkan pemilihan simpul dengan nilai f(n) terkecil dan ekspansi simpul tersebut memastikan bahwa jalur yang ditemukan tidak hanya terpendek tetapi juga optimal. Dalam contoh yang diberikan, jalur optimal yang ditemukan adalah A -> B -> C -> D -> J -> K -> I -> Q -> R -> S.

5. Fleksibilitas Algoritma, algoritma A* menunjukkan fleksibilitas yang tinggi dalam penyesuaian terhadap berbagai ukuran dan bentuk labirin. Perubahan atau pembaruan dalam struktur labirin, seperti penambahan atau penghapusan rintangan, dapat dilakukan dengan mudah tanpa mempengaruhi efektivitas dan efisiensi algoritma.

Secara keseluruhan, implementasi algoritma A* dalam pemecahan masalah pencarian jalur terpendek pada labirin memberikan hasil yang memuaskan. Algoritma ini dapat diandalkan untuk menyelesaikan berbagai jenis masalah pencarian jalur, baik dalam skala kecil maupun besar, dengan tetap mempertahankan efisiensi dan efektivitasnya. Dengan demikian, implementasi algoritma A* pada labirin tidak hanya memberikan solusi yang optimal untuk masalah pencarian jalur, tetapi juga memberikan bukti kuat akan kegunaan praktis algoritma ini dalam berbagai aplikasi di dunia nyata.

UCAPAN TERIMA KASIH

Dengan penuh rasa syukur, penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas segala anugerah dan kasih sayang-Nya yang tiada henti. Berkat rahmat dan karunia-Nya, penulis dapat menyelesaikan tulisan yang berjudul "Optimalisasi Pencarian Jalur dalam Labirin Menggunakan Algoritma A*.". Tidak lupa, terima kasih kepada keluarga yang memberikan dorongan dan dukungan selama proses penulisan. Penghargaan juga disampaikan kepada Ir. Rila Mandala, M.Eng., Ph.D., dan Monterico Adrian, S.T., M.T., sebagai dosen pembimbing Mata Kuliah Strategi Algoritma ITB Kelas 3, atas panduan dan ilmu yang telah dibagikan, sehingga penulis dapat meyelesaikan pembuatan makalah ini. Akhir kata, penulis berharap semoga makalah ini dapat memberikan kontribusi yang berarti dalam bidang ilmu komputer, khususnya dalam penerapan algoritma pencarian jalur. Penulis juga berharap semoga bimbingan dan dukungan yang telah diberikan oleh semua pihak dapat memberikan manfaat yang berkelanjutan dan menjadi berkah bagi kita semua.

REFERENSI

[1] R. Munir, "Penentuan Rute (Route/Path Planning) (Bag.1). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf, pada 10 Juni 2024.

[2] R. Munir, "Penentuan Rute (Route/Path Planning) (Bag.2). Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf, pada 10 Juni 2024.

[3] A-Star (A*) Search for Solving a Maze using Python (with visualization). Diakses dari https://levelup.gitconnected.com/a-star-a-search-for-solving-a-maze-using-python-with-visualization-b0cae1c3ba92, pada 11 Juni 2024.

[4] A* Pathfinding Algorithm: Efficiently Navigating the Maze of Possibilities. Diakses dari https://panda-man.medium.com/a-pathfinding-algorithm-efficiently-navigating-the-maze-of-possibilities-8bb16f9cecbd, pada 11 Juni 2024.

[5] Penerapan Algoritma A* (A Star) Sebagai Solusi Pencarian Rute Terpendek Pada Maze. Diakses dari https://www.researchgate.net/publication/309562971_Penerapan_Algoritma_A_A_Star_Sebagai_Solusi_Pencarian_Rute_Terpendek_Pada_Maze, pada 11 Juni 2024.

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.

Bandung, 12 Juni 2024

Muhammad Fatihul Irhab 13522143