e-ISSN: 2549-7952
p-ISSN: 2580-3336

Implementasi Algoritma Backtracking untuk Mencari Jalan Keluar Labirin

Yosafat Adiguna[1], Daniel Swanjaya[2]
[1,2]Teknik Informatika, Fakultas Teknik, Universitas Nusantara PGRI Kediri
E-mail: [1]*yosafatadiguna30@gmail.com*, [2]*daniel@unpdkediri.ac.id*

Abstrak – *Labirin merupakan teka- teki yang sering kali menjadi sebuah permainan/game. Labirin memiliki banyak cabang dan jalan buntu yang membuat labirin sulit untuk dicari jalan keluarnya. Algoritma pencarian dapat memudahkan pencarian jalan keluar sebuah labirn, namun tidak semua algoritma pencarian dapat diimplementasikan pada sebuah labirin. Oleh karena itu menentukan algoritma yang tepat menjadi salah satu faktor penentu untuk mencari jalan keluar sebuah labirin. Algoritma Backtracking merupakan salah satu dari algoritma pencarian yang memiliki tingkat efisiensi yang tinggi karena Backtracking merunut balik dari simpul tujuan untuk melihat apakah solusi yang sedang dicari menuju pada simpul tujuan yang diinginkan. Dengan demikian algoritma Backtracking dapat memangkas langkah-langkah yang tidak perlu dalam sebuah pencarian dan dapat mencari rute terpendek dalam sebuah pencarian. Dari sisi tersebut algoritma Backtracking memiliki kelebihan dibandingan algoritma Depth First Search yang tidak mempertimbangkkan apakah solusi yang sedang dicari menuju pada titik tujuan yang diinginkan. Penelitian ini dilakukan dengan menggunakan labirin berupa matriks dengan ukuran n x n dan menggunakan algoritma Backtracking untuk mencari jalan keluar dari labirin, hasil yang didapat dalam penelitian ini berupa jumlah langkah dan simpul- simpul yang dilewati selama pencarian sampai menuju simpul tujuan.*

Kata Kunci *— Backtracking, implementasi, labirin*

1. PENDAHULUAN

Menurut KBBI labirin merupakan tempat yang penuh dengan lorong dan jalan yang berliku-liku dan simpang siur [1]. Menurut Lexico labirin adalah sebuah jaringan dari jalan-jalan atau batas-batas yang dibuat sebagai puzzle (teka-teki) yang akan dicari jalan keluarnya [2]. Menurut Cambridge Dictionary (Kamus Cambridge) labirin adalah kumpulan bagian-bagian atau jalan-jalan yang saling berhubungan dan membingungkan yang adalah mudah untuk tersesat di dalamnya [3]. Dari definisi-definisi diatas dapat disimpulkan bahwa labirin adalah tempat yang berliku-liku dan membingungkan.

Labirin dapat memiliki jalan bercabang yang bukan merupakan rute tercepatnya dan jalan buntu yang bukan merupakan jalan keluar. Dengan permasalahan tersebut pencarian jalan keluar dengan menggunakan algoritma pencarian dapat menemui hasil yang kurang optimal seperti rute yang di dapat bukanlah rute terpendek. Dengan menggunakan algoritma *Backtracking* dimungkinkan untuk mencari rute terpendek karena algoritma *Backtracking* melakukan runut balik untuk melihat apakah solusi yang sekarang dikerjakan menuju ke solusi yang diinginkan atau tidak, dengan demikian memperpendek rute pencarian yang mungkin terjadi.

Seperti penelitian yang dilakukan oleh Imam Ahmad dan Wahyu Widodo yang menggunakan algoritma *A** untuk mencari rute terpendek sebuah labirin *game* labirin. Dengan menggunakan algoritma *A** dengan fungsi pencarian *heuristic* menggunakan *Euclidian Distance* rute terpendek dalam sebuah labirin dapat ditemukan [4]. Berbeda dengan algoritma *A** algoritma *Backtracking* tidak melakukan pencarian *heuristic* namun mempertimbangkan apakah solusi jalan yang ditempuh mengarah kepada tujuan.

Penelitian lain dilakukan oleh Sayed Fachrurrazi yang menggunakan algoritma Prim untuk mencari rute terpendek jalan keluar sebuah labirin. Algoritma Prim mencari *minimum spanning tree* (pohon perentangan minimum) dengan menggunakan graf berbobot yang dimulai secara acak lalu mencari nilai terkecilnya [5]. Berbeda dengan algoritma *Backtracking* yang dimulai di simpul akar dan mencari seluruh kemungkinan simpul yang ada dengan mempertimbangkan apakah simpul solusi yang ada mengarah ke simpul tujuan yang diinginkan.

Penelitian menggunakan algoritma *Breadth First Search* (BFS) untuk mencari rute terpendek jalan keluar sebuah labirin dalam sebuah *game* berbasis android pernah dilakukan oleh Astrid Novita Putri. Penelitian ini dilakukan dengan cara menggunakan algoritma BFS untuk membantu *player* menuju jalan keluar dengan lebih cepat [6]. Algoritma BFS bekerja dengan cara memeriksa seluruh simpul anak yang ada dan mengulang proses tersebut sampai menemukan simpul tujuan.

Algoritma Dijkstra digunakan oleh Yusuf Anshori, A. Y. Erwin Dodu dan Fadli Kurniawan untuk diterapkan pada robot yang digunakan untuk mencari rute terpendek jalan keluar labirin yang sudah diidentifikasi oleh robot [7]. Algoritma ini bekerja dengan graf berbobot dan mempertimbangkan bobot terkecil dari simpul akar sampai simpul daun tujuan. Dengan demikian didapati rute terpendek.

Penelitian yang dilakukan oleh Mustika Rahmawati, Husnul Hotimatus S., dan Maulia Azizah menggunakan algoritma DFS untuk mencari jalan keluar sebuah labirin dengan waktu tercepat.

e-ISSN: 2549-7952
p-ISSN: 2580-3336

Algoritma ini memiliki kelemahan untuk mencari rute terpendek karena algoritma ini hanya menelusuri simpul-simpul solusi sampai pada simpul daun dan melakukan runut balik jika simpul daun bukan simpul daun yang diinginkan [8]. Algoritma DFS melihat apakah simpul-simpul solusi mengarah ke simpul tujuan sehingga akan berpengaruh pada rute yang ditempuh.

2. METODE PENELITIAN

2.1 *Tree*

Tree (pohon) secara rekursif dapat diartikan sebagai satu atau lebih kumpulan simpul yang salah satu simpulnya merupakan akar sedangkan simpul-simpul yang lain dapat menjadi kumpulan-kumpulan simpul yang masing-masing adalah sub-tree(bagian kecil pohon) dari akar. Gambar 1 adalah gambar *tree* [9].

Terminologi dasar sebuah *tree/* pohon:

1. Simpul akar (*root node*) adalah simpul yang berada di tempat paling atas sebuah *tree.* Jika simpul akar bernilai *null* maka *tree* yang ada kosong.
2. *Sub-tree*, jika simpul akar tidak bernilai *null* maka T_1, T_2 dan T_3 adalah *sub-tree* dari simpul akar.
3. Jalan (*Path)* adalah serangkaian sisi yang saling berhubungan.
4. Simpul orang tua *(Ancestor node*) adalah simpul manapun yang menjadi simpul pendahulu dari simpul akar akan menuju simpul.
5. Simpul anak (*Descendant node*) adalah simpul manapun yang menjadi simpul penerus dari simpul akar sampai simpul sekarang.
6. Nomer Level, setiap simpul dalam *tree* memiliki nomer level sedemikian rupa sehingga nomer level simpul akar adalah nol, simpul anak dari simpul akar bernilai 1. Lalu setiap simpul memilki nomer level yang lebih tinggi daripada simpul orang tuanya.
7. Derajat, derajat sebuah simpul adalah jumlah anak simpul yang dimiliki oleh simpul tersebut. Nilai derajat simpul daun adalah nol.

   Adapun jenis-jenis *Tree* adalah:

   1. *Tree*
   2. Hutan (*Forest*)
   3. *Pohon biner (Binary Tree*)
   4. Pohon Pencarian Biner (*Binary Search Tree*)
   5. Pohon Ekspresi (*Expression Tree*)
   6. Pohon Turnamen (*Tournament Tree*)

2.2 Graf

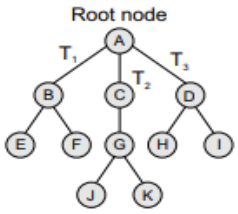
Graf adalah sebuah struktur data yang abstrak yang digunakan untuk mengimplementasikan konsep matematis sebuah graf. Sebuah graf sering dipandang sebagai generalisasi dari sebuah *tree*/ pohon dimana daripada memiliki hubungan simpul *parent – child tree* graf dapat memiliki hubungan kompleks antar simpul yang dapat terjadi [10].

Terminologi dasar dalam Graf:

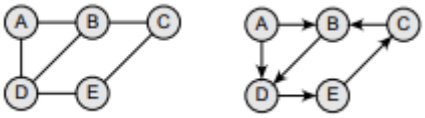
1. Tetangga (*neighbours*), untuk setiap sisi , e = (v, u) yang menghubungkan simpul u dengan simpul v, simpul v dan simpul u berdekatan atau bertetanggaan.
2. Derajat simpul (*degree of nodes*) adalah banyaknya sisi yang tersambung pada sebuah simpul.
3. Graf Reguler (*Regular Graph*) adalah graf di mana setiap simpulnya memiliki jumlah tetangga yang sama, dengan kata lain derajat simpul dalam sebuah graf reguler adalah sama.
4. Jalan (*Path)* adalah serangkaian sisi yang saling berhubungan.
5. Graf Tak Berarah *(Undirected Graph*) adalah sebuah graf yang sisi nya tidak memiliki arah.
6. Graf Berarah (*Directed Graph*) adalah sebuah graf yang sisi nya memiliki arah.

Sebuah graf dapat bearah atau tidak berarah, pada graf tidak berarah sisi tidak mempunyai arah manapun yang terasosikan dengan sisi tersebut. Oleh karena itu jika digambar diantara simpul A dan B sehingga simpul dapat mengarah dari A ke B atau sebaliknya dari B ke A. Gambar 2.a adalah gambar graf tak berarah.

Dalam Graf Berarah sisi membentuk sebuah pasangan yang berurutan. Terdapat sebuah jalan yang berasal dari A menuju B dan tidak dari B ke A. sisi (A, B) diinisialisasikan dari simpul A(disebut sebagai simpul awal) dan berhenti di simpul B (disebut sebagai simpul terminal). Gambar 2.b adalah gambar graf berarah sederhana.



Gambar 1. Tree



(a) (b)

Gambar 2. (a) Graf Tak Berarah. (b) Graf Berarah

2.3 Algoritma *Backtracking*

Backtracking adalah cara sistematis untuk mengulang seluruh konfigurasi yang mungkin terjadi dari sebuah ruang pencarian. Konfigurasi ini dapat berupa semua susunan permutasi objek yang dapat terjadi atau seluruh cara yang mungkin untuk membuat kumpulan dari objek-objek tersebut [11].

Backtracking adalah algoritma yang berbasis pada *Depth First Search* (DFS) untuk mencari solusi persoalan lebih cepat. Runut balik yang merupakan perbaikan dari algoritma *brute-force*, secara sistematis mencari solusi persoalan di antara semua

e-ISSN: 2549-7952
p-ISSN: 2580-3336

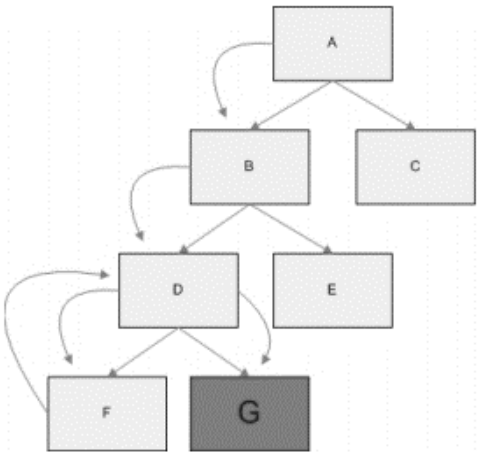
kemungkinan solusi yang ada. Dengan langkah ini tidak perlu memeriksa semua kemungkinan solusi yang ada. Akibatnya waktu pencarian dapat dihemat. *Backtracking* lebih alami disebut sebagai algoritma *rekursif*. Kadang disebutkan pula bahwa *Backtracking* merupakan bentuk tipikal dari algoritma *rekursif*. *Backtracking* pertama kali dikenalkan oleh D. H. Lehmer pada tahun 1950. R. J. Walker, Golomb, Baumert menyajikan uraian umum tentang *Backtracking* dan penerapannya pada berbagai persoalan [12].

Pencarian dengan menggunakan algoritma *Backtracking* mempunyai langkah-langkah sebagai berikut:

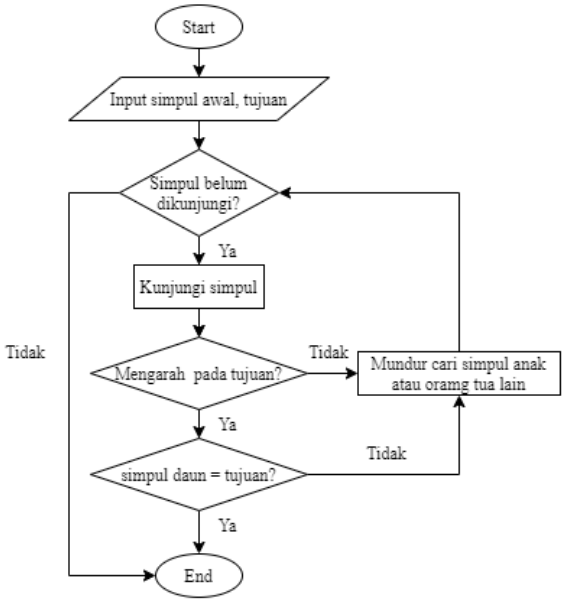
1. Solusi didapat dengan membentuk lintasan dari simpul akar sampai simpul daun. Simpul yang dilahirkan dinamakan simpul hidup dan simpul hidup yang diperluas dinakan simpul-E (*Expand node*).
2. Jika lintasan yang diperoleh dari perluasan simpul-E tidak mengarah pada solusi, maka simpul itu tidak akan diperluas lagi.
3. Jika posisi terakhir ada disimpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lainnya dan jika tidak ada simpul anak maka dilakukan *backtracking* ke simpul orang tua.
4. Pencarian dihentikan ketika menemukan solusi atau tidak ada simpul hidupyang dapat diperluas.

Gambar 4 adalah gambaran cara kerja a*lgoritma Backtracking*. Pada Gambar 4 simpul akar adalah A dan simpul tujuan adalah G. Pencarian dimulai dari memeriksa simpul anak dari simpul A yang adalah simpul B dan C. Lalu simpul B dipilih karena menuju pada simpul tujuan. Lalu diperiksa simpul anak dari simpul B yaitu D dan E. Simpul D lalu dipilih Karena mengarah pada simpul tujuan. Lalu diperiksa simpul anak dari D yaitu F dan G. lalu simpul F secara *default*, karena simpul F adalah simpul daun maka diperiksa apakah simpul F merupakan tujuan dan karena F bukan tujuan maka pencarian dilakukan dengan *backtrack* ke simpul D lalu memilih simpul anak lain dari simpul D. Simpul G lalu dipilih dan diperiksa apakah simpul G merupakan tujuan, karena simpul G merupakan simpul tujuan pencarian selesai.

Gambar 5 adalah gambar flowchart algoritma *Backtracking* untuk mencari jalan keluar sebuah labirin. Langkah pertama yang dilakukan dalam algoritma *Backtracking* seperti yang terlihat dalam gambar 5 adalah memasukkan simpul awal dan simpul tujuan. Lalu dilakukan pengecekan apakah ada simpul yang belum dikunjungi dari simpul aktif jika tidak maka pencarian dihentikan. Jika ada yang belum dikunjungi maka simpul dikunjungi lalu dilakukan pengecekan apakah jalan yang ditempuh mengarah pada simpul tujuan, jika tidak maka dilakukan *backtrack* ke simpul anak atau orang tua lain lalu pencarian dilanjutkan dari langkah pengecekan simpul yang belum dikunjungi. Jika simpul menuju pada simpul tujuan lalu dilakukan proses perulangan dengan memilih simpul yang sebagai simpul aktif. Jika mencapai simpul daun maka dilakukan pengecekan apakah simpul daun yang didapat adalah simpul tujuan dan jika tidak maka dilakukan *backtrack*. Jika simpul daun adalah tujuan maka pencarian selesai.



Gambar 4. Contoh Algoritma *Backtracking*



Gambar 5. Flowchart Algoritma *Backtracking*

3. HASIL DAN PEMBAHASAN

3.1. Kebutuhan Data

Kebutuhan data yang diperlukan dalam penelitian ini adalah data berupa labirin dalam bentuk matriks. Data tersebut di dapat dari [13].

Contoh matriks labirin yang digunakan dalam penelitian ini adalah matriks seperti pada gambar 6.

Gambar 6 merupakan matriks labirin yang akan digunakan untuk penelitian ini, matriks ini berisi angka 1 dan 0. Angka 1 mewakili jalan yang dapat dilalui sedangkan angka 0 mewakili tembok yang tidak dapat dilalui. Matriks memiliki ukuran 12 x 12, dengan 12 baris yang memiliki indeks 0 sampai 11 dimulai dari kiri atas dan 12 kolom dengan jumlah indeks yang sama.

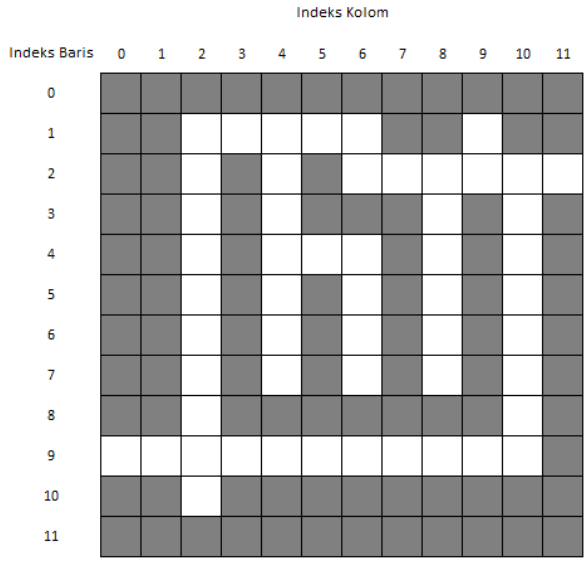
e-ISSN: 2549-7952
p-ISSN: 2580-3336

Gambar 7 adalah gambar visualisasi matriks labirin dari gambar 6.

Indeks kolom

Indeks baris	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	0	0	1	**0**	0
2	0	0	1	0	1	0	1	1	1	1	1	1
3	0	0	1	0	1	0	0	0	1	0	1	0
4	0	0	1	0	1	1	1	0	1	0	1	0
5	0	0	1	0	1	0	1	0	1	0	1	0
6	0	0	1	0	1	0	1	0	1	0	1	0
7	0	0	1	1	1	0	1	0	1	0	1	0
8	0	0	1	0	0	0	0	0	0	0	1	0
9	1	1	1	1	1	1	1	1	1	1	1	0
10	0	0	1	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0

Gambar 6. Contoh Matriks Labirin



Gambar 7. Visualisasi Contoh Matriks Labirin

3.2. Hasil Testing

Dengan menggunakan matriks labirin contoh kasus yang terdapat diatas dilakukan pencarian jalan keluar dengan menggunakan algoritma *Backtracking*.

Dengan simpul awal dari kasus tersebut adalah koordinat (9,0) dan simpul tujuan dari kasus tersebut adalah (2,11). Dari hasil percobaan ditemukan bahwa algoritma *Backtracking* berhasil menemukan jalan keluar pada labirin ini dan jalan keluar yang ditemukan merupakan rute tercepat. Hasil pencarian pada contoh kasus yang dipakai dapat dilihat dalam tabel 1.

Dari hasil pencarian pada tabel 1 didapat bahwa untuk mencari jalan keluar pada contoh kasus diperlukan 19 langkah dan jalan keluar yang didapat merupakan rute terpendek. Dari percobaan juga didapat bahwa durasi pencarian adalah 6 milisekon. Dari data hasil pencarian ada beberapa hal yang perlu di perhatikan yaitu bahwa algoritma *Backtracking* mampu mencari rute terpendek saat pencarian labirin berlangsung. Dapat dilihat pada iterasi ke 4 di Tabel 1 bahwa setelah dari simpul (9,2) perulangan menuju (9,3) dan bukan menuju (10,2). Hal ini dapat terjadi karena algoritma *Backtracking* mempertimbangkan apakah simpul (10,2) mengarah pada simpul tujuan dan jawabannya adalah tidak, sehingga simpul (10.2) dimatikan dan tidak dilewati. Jika algoritma yang digunakan pada kasus ini adalah algoritma DFS maka (10,2) akan dilewati sehingga jumlah langkah menjadi bertambah karena algoritma DFS mencari solusi dengan mencoba semua kemungkinan yang dapat terjadi dari graf dan mencari sampai ke simpul daun.

Tabel 1. Data Pencarian Jalan Keluar Pada Labirin

Iterasi ke-	Lintasan Yang Di Lalui	Jumlah Langkah
1	(9,0)	1
2	(9,0).(9,1)	2
3	(9,0).(9,1),(9,2)	3
4	(9,0).(9,1),(9,2),(9,3)	4
5	(9,0).(9,1),(9,3),(9,4)	5
6	(9,0).(9,1),(9,3),(9,4),(9,5)	6
7	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6)	7
8	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6),(9,7)	8
9	(9,0).(9,1),(9,3),(9,4), (9,5),(9,6),(9,7),(9,8)	9
10	(9,0).(9,1),(9,3),(9,4),(9,5), (9,6),(9,7),(9,8),(9,9)	10
11	(9,0).(9,1),(9,3),(9,4),(9,5), (9,6),(9,7),(9,8),(9,9),(9,10)	11
12	(9,0).(9,1),(9,3),(9,4),(9,5), (9,6),(9,7),(9,8),(9,9),(9,10),(8,10)	12
13	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6),(9,7), (9,8),(9,9),(9,10),(8,10),(7,10)	13
14	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6),(9,7), (9,8),(9,9),(9,10),(8,10),(7,10),(6,10)	14
15	(9,0).(9,1),(9,3),(9,4),(9,5), (9,6),(9,7),(9,8),(9,9),(9,10), (8,10),(7,10),(6,10),(5,10)	15
16	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6), (9,7),(9,8),(9,9),(9,10),(8,10), (7,10),(6,10),(5,10),(4,10)	16
17	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6), (9,7),(9,8),(9,9),(9,10),(8,10), (7,10),(6,10),(5,10),(4,10),(3,10)	17
18	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6),(9,7), (9,8),(9,9),(9,10),(8,10),(7,10), (6,10),(5,10),(4,10),(3,10),(2,10)	18
19	(9,0).(9,1),(9,3),(9,4),(9,5),(9,6),(9,7), (9,8),(9,9),(9,10),(8,10),(7,10),(6,10), (5,10),(4,10),(3,10),(2,10),(2,11)	19
END		

e-ISSN: 2549-7952
p-ISSN: 2580-3336

Tabel 3 adalah tabel dari 10 percobaan penggunaan algoritma *Backtracking* untuk labirin-labirin yang berbeda:

Tabel 3. Hasil Percobaan

Simpul Awal	Simpul Tujuan	Jumlah Langkah	Lintasan Terpendek	Durasi (ms)
(9,0)	(2,11)	19	Ya	62
(8,0)	(3,11)	21	Tidak	47
(8,0)	(1,11)	23	Tidak	8
(8,0)	(1,11)	21	Ya	0
(9,0)	(2,11)	23	Tidak	23
(10,0)	(2,11)	21	Ya	0
(8,0)	(1,11)	24	Ya	16
(7,0)	(1,11)	19	Ya	0
(8,0)	(2,11)	13	Ya	47
(6,0)	(7,11)	26	Tidak	0

3.3. Pembahasan

Dari kesepuluh labirin terdapat hasil yang berbeda-beda. Di setiap labirin memiliki simpul awal, simpul tujuan, jumlah langkah, durasi waktu dan rute solusi yang berbeda. Hal ini dapat terjadi karena setiap labirin memiliki isi yang berbeda.

Durasi waktu yang bermacam-macam terjadi juga karena setiap labirin yang digunakan memiliki isi berbeda, meskipun simpul awal dan simpul akhir yang dimiliki dua labirin sama. Durasi waktu didapat dengan mengurangkan nilai waktu setelah algoritma selesai dengan nilai waktu saat algoritma dimulai. Satuan durasi waktu yang dipakai adalah milisekon. Dalam Tabel 4 terdapat beberapa durasi waktu yang bernilai 0 milisekon, artinya proses pencarian jalan keluar dengan algoritma tidak mencapai durasi 1 milisekon atau nilai waktu setelah algoritma selesai dikurangi sebelum algoritma mulai tidak mencapai 1 milisekon.

Lintasan terpendek dari masing-masing labirin didapat dari hasil perhitungan manual yang dilakukan sebelum algoritma dijalankan. Dari 10 percobaan 4 kali tidak menghasilkan rute terpendek dan 6 kali menghasilkan rute terpendek. Hal ini disebabkan oleh isi labirin yang berbeda-beda, sehingga untuk menemukan rute terpendek dalam sebuah labirin menggunakan algoritma *Backtracking* hasilnya akan sangat bergantung pada isi dari labirin yang akan dicari jalan keluarnya.

4. SIMPULAN

Dari 10 labirin yang digunakan untuk pencarian jalan keluar dengan menggunakan algoritma *Backtracking* dapat disimpulkan bahwa algoritma *Backtracking* dapat diimplementasikan untuk mencari jalan keluar sebuah labirin.

Algoritma *Backtracking* dapat digunakan mencari rute terpendek jalan keluar sebuah labirin, namun hal ini sangat bergantung kepada isi dari labirin yang akan dicar jalan keluarnya.

Algoritma *Backtracking* memiliki efisiensi yang tinggi karena mempertimbangkan rute yang sedang dicari apakah akan menuju simpul yang diinginkan atau tidak sehingga pencarian tidak perlu dilakukan dengan mencari semua kemungkinan yang dapat terjadi, dengan demikian mempercepat proses pencarian dan mengurangi simpul-simpul yang harus dicari.

Algoritma *Backtracking* memiliki keunggulan dibandingkan dengan algoritma pencarian DFS yang menelusuri semua kemungkinan yang dapat terjadi sehingga membuat algoritma tidak efisien dan memerlukan banyak sumber daya.

Algoritma *Backtracking* lebih sederhana dibandingankan dengan algoritma pencari rute terpendek seperti A*. Algoritma A* dapat mencari rute terpendek dengan menggunakan *open* dan *closed list.*

5. SARAN

Algoritma *Backtracking* meskipun sudah mempertimbangkan rute yang sedang dikerjakan tetap mungkin mengambil rute yang bukan rute menuju simpul tujuan. Kiranya penelitian selanjutnya dapat menggunakan algoritma yang dapat menyelesaikan masalah ini.

Labirin yang digunakan dalam penelitian ini adalah matriks 2 dimensi berukuran n x n, untuk pengembangan penelitian ke depan dapat menggunakan matriks 3 dimensi yang berukuran n x n x n.

DAFTAR PUSTAKA

[1] Badan Pengembangan Bahasa dan Perbukuan, Kementrian Pendidikan dan Kebudayaan Republik Indonesia. 2016. https://kbbi.kemdikbud.go.id/entri/labirin diakses pada 23 Februari 2020.

[2] Dictionary.com dan Oxford University Press. 2020. https://www.lexico.com/en/definition/labyrinth diakses pada 23 Februari 2020.

[3] Cambridge University Press. 2020. https://dictionary.cambridge.org/dictionary/english/labyrinth diakses pada 23 Februari 2020.

[4] Ahmad, I., Widodo, W., 2017. Penerapan Algoritma A Star (A*) Pada Game Petualangan Labirin Berbasis Android. *Jurnal Ilmu Komputer dan Informatika*. Vol. III, No. 2: 57-63.

[5] Fachrurrazi, S., 2018. Sistem Penentuan Rute Yang Tepat Dalam Sebuah Labirin Dengan Menerapkan Algoritma Prim. *Jurnal Sistem Informasi*. Vol. II, Np. 1: 51-67.

[6] Putri, N. A., 2016. Optimasi Alghoritma Breadth First Search Pada Game Engine Third Person Shooter Maze Berbasis Agen Cerdas Android. *Jurnal Transformatika*. Vol. XIV, No. 1: 50-55.

[7] Anshori, Y., Dodu, E. A. Y., Kurniawan, F. Perancangan Robot Penelusur Menggunakan Algoritma Dijkstra Dan Metode Maze Solver. *Techno.COM*. Vol. XVIII, No. 2: 166-177.

[8] Azizah, M., Hostimatus S., H., Rahmasuci, M., 2018, Strategi Menemukan Jalan Keluar Dengan Waktu Tercepat Menggunakan Metode DFS. *Informatic Journal*. Vol. III, No. 1: 12-16.

e-ISSN: 2549-7952
p-ISSN: 2580-3336

[9] Thareja, R. 2014. DATA STRUCTURES USING C. India: Oxford Unviersity Press. Hal. 279.
[10] Thareja, R. 2014. DATA STRUCTURES USING C. India: Oxford Unviersity Press. Hal. 383.
[11] Skiena, S. 2008. The Algorithm Design Manual. New York: Springer. Hal. 271.
[12]` Siraif, Br. Rina., 2013. Perancangan Aplikasi Game Labirin Dengan Menggunakan Algoritma Backtracking. *Pelita Informasi Budi Darma*. Vol. V, N0. 2: 101-103.
[13] https://github.com/AkshayGuptaK/maze-solvers/blob/master/Maze%20Solver%20Step%20Through.xlsx.