Penyelesaian *Maze Problem* Dengan Berbagai Algoritma *Pathfinding*

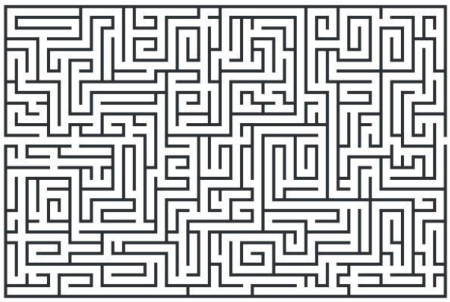
Roby Purnomo - 13520106
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): robypurnom@gmail.com

***Abstraksi*—*Maze* atau *Labyrinth* telah menjadi persoalan sejak zaman dulu kala hingga sekarang. Banyak sekali permainan/*game*, film, bahkan di dunia nyata sekalipun yang mengadaptasi konsep *puzzle* ini. *Maze* ditujukan agar penelusur susah menemukan jalan akhir dengan cepat, yang pada intinya memperpanjang lintasan yang dilewati oleh penelusur. Dengan seiringnya berkembangnya zaman yang serba memudahkan segalanya, *Maze* Problem ini pun juga dimudahkan penyelesaiannya yakni beberapa diantaranya dengan menggunakan berbagai algoritma pencarian rute seperti menggunakan DFS, BFS, dan A*.**

Keywords—maze; labyrinth; puzzle; game

I. PENDAHULUAN

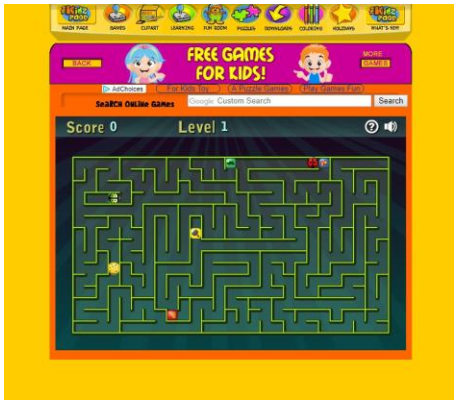
Maze atau *Labyrinth* yang memiliki arti pada bahasa Indonesia yakni labirin adalah sebuah kata yang muncul sejak Plutarch menulis tulisan mengenai "*Myth of the Minotaur*". Labirin adalah sebuah ruang yang memiliki batas-batas yang harus dilalui dengan tujuan utama untuk menghambat siapa saja yang melewatinya. Menghambat disini maksudnya adalah menunda perjalanan sehingga menempuh waktu yang lebih lama. Perjalanan dapat ditunda dengan berbagai cara, salah satunya adalah dengan memperpanjang lintasan target, hal inilah yang diadaptasi pada konsep labirin. Labirin dibuat seakan-akan penelusur harus melewati jalan yang berkelok-kelok, rumit, dan memiliki jalan buntu sehingga harus memutar sehingga menambah jarak tempuh bagi si penelusur labirin.



Gambar 1. Labirin

(sumber : https://www.istockphoto.com/id/ilustrasi/maze)

Di masa sekarang ini, Maze atau Labyrinth sudah menjadi kata yang sangat umum didengar oleh semua kalangan. Mulai dari permainan, film, bahkan di dunia nyata sekalipun terdapat labirin yang biasanya dijadikan sebuah puzzle. Contohnya saja dalam permainan Gadget yang terdapat pada Gambar 2 dibawah ini. Selain itu, juga banyak permainan yang mengadaptasi *Maze Problem* ini ke dalam salah satu *mini quest* mereka untuk melengkapi berbagai tantangan.



Gambar 2. Permainan Gadget tentang *Maze Problem*

(sumber : https://www.thesprucecrafts.com/free-online-mazes-1357461)

Pada makalah ini, penulis akan membahas mengenai beberapa algoritma yang dapat digunakan untuk memecahkan permasalahan labirin. Beberapa algoritma yang akan digunakan adalah BFS (Breadth First Search), DFS (Depth First Search), dan algoritma A*.

II. TEORI DASAR

A. *BFS (Breadth First Search)*

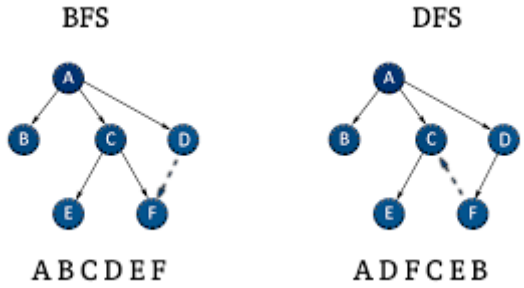
Algoritma BFS diusulkan secara resmi pada tahun 1959 oleh E. F. Moore dalam konteks pencarian jalur pada labirin dan pada tahun 1961, Lee secara independen menemukan algoritma yang sama padaa konteks yang berbeda yakni konduktor PCB. Algoritma BFS memungkinkan pencarian jarak terpendek dari simpul yang dipilih dari graf berarah ke semua simpul lainnya. Algoritma ini juga dapat digunakan untuk menemukan jalur terpendek antara dua simpul.

Algoritma BFS mengadopsi konsep FIFO (First-In-First-Out) atau queue. Algoritma ini dapat dijelaskan ibarat melintasi simpul layer per layer, jadi arah dari algoritma ini adalah berdasarkan lapisan. Pada setiap langkah, akan dicek mana saja

simpul yang belum dicek, jika belum dicek maka dimasukkan pada queue untuk dicek node yang bertetangga dengannya.

B. DFS (Depth First Search)

Algoritma DFS telah ada sejak abad ke-19 ditemukan oleh matematikawan Prancis yakni Charles Perre Tremo. Algoritma Tremo adalah algoritma yang didesain efektif untuk menemukan jalan keluar dari sebuah labirin dengan cara menandai jalan yang menuju jalan keluar dari labirin dari jalan masuk labirin. Namun, baru 100 tahun kemudian algoritma ini digunakan sebagai dasar dari algoritma Depth First Searching atau DFS. Algoritma ini cukup sederhana yakni dari simpul awal lalu secara acak (atau jika dengan heuristik dapat menambahkan fungsi pembangkit) memilih salah satu simpul tetangga lalu dari simpul tetangga ini akan melakukan hal yang sama hingga tidak ada jalan lagi, barulah menginjak ke simpul selanjutnya hingga akhir. Algoritma ini menggunakan konsep LIFO (Last-In-First-Out) atau stack dan mengadaptasi konsep rekursif pada algoritmanya. Algoritma ini sangat sederhana tetapi masalahnya adalah perhitungan perpangkatan yang besar sehingga seiring bertambahnya ukuran graf maka kompleksitasnya akan bertambah secara eksponensial.



Gambar 3. BFS dan DFS

(sumber : https://open4tech.com/bfs-vs-dfs/)

*C. Algoritma A**

Algoritma A* adalah sebuah generic search algorithm yang dapat digunakan untuk menemukan solusi untuk beberapa masalah salah satunya adalah pencarian jalur. Algoritma ini merupakan gabungan dari algoritma uniform-cost search dan heuristic search. Dalam menyelesaikan permasalahan pencarian jalur atau pathfinding, algoritma A* akan berulang kali memeriksa lokasi yang paling bernilai untuk dikunjungi. Dalam kasus yang lain, algoritma ini juga membantu mencatat semua tetangga simpul/lokasi sebagai eksplorasi tambahan. A* merupakan algoritma pathfinding yang paling populer pada game AI. Kompleksitas dari algoritma ini tergantung pada fungsi heuristik yang digunakan pada programnya.

III. ALGORITMA

A. BFS (Breadth First Search)

Pseudocode algoritma BFS secara umum adalah sebagai berikut.

```
procedure BFS (input n : integer, input goal : integer)
{
    Traversal pencarian pada graf dengan menggunakan algoritma BFS

    input : n adalah simpul awal
    output : mengembalikan rute node awal ke node goal
}

DEKLARASI
    q : queue    # queue sebagai buffer untuk simpul
    s : integer  # simpul yang ditrack
    visited : array of boolean

    procedure Queue(input/output q : queue)
    { Konstruktor queue }

    procedure push(input/output q : queue, input n : integer)
    { insert integer n pada queue sebagai tail
}

    procedure pop(input/output q : queue, output n : integer)
    { delete head dari queue }

    function isEmpty(input q : queue) -> boolean
    { mengecek apakah queue kosong atau tidak }

    function nodeAdjacent(input n : integer) -> array of integer
    { mengembalikan array node yang bertetangga dengan node n }

    function traceBackward(input n : integer, input goal : integer) -> array of integer
    { mengembalikan array node rute dari node awal menuju node goal }


ALGORITMA
    Queue(q)
    visited[n] <- true
    push(q, n)

    while not isEmpty(q) do
        pop(q, n)
        for s in nodeAdjacent(n) do
            if not visited[s] then
                if (s = goal) then
                    traceBackward(n, s)
                else
                    push(q, s)
                    visited[s] <- true
                endif
            endif
        endfor
    endwhile
```

B. DFS (Depth First Search)

Pseudocode algoritma BFS secara umum adalah sebagai berikut.

```
procedure DFS (input n : integer, input goal :
integer)
{
    Traversal pencarian pada graf dengan
menggunakan algoritma DFS

    input : n adalah simpul awal
    output : mengembalikan rute node awal ke
node goal
}

DEKLARASI
    s : integer  # simpul yang ditrack
    visited : array of boolean

    function nodeAdjacent(input n : integer) ->
array of integer
    { mengembalikan array node yang bertetangga
dengan node n }

    function traceBackward(input n : integer,
input goal : integer) -> array of integer
    { mengembalikan array node rute dari node
awal menuju node goal }

ALGORITMA
    visited[n] <- true

    for s in nodeAdjacent(n) do
        if not visited[s] then
            if (s = goal) then
                traceBackward(n, s)
            else
                DFS(s)
            endif
        endif
    endfor
```

*C. Algoritma A**

Sedangkan untuk algoritma A* memiliki langkah-langkah sebagai berikut :

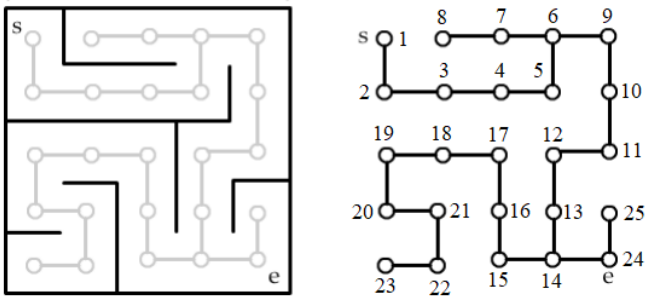
1. Variabel n adalah simpul awal
2. Masukkan nilai g, h, dan f ke simpul n

g	Cost yang didapat dari simpul awal menuju simpul saat ini
h	Estimasi cost dari simpul saat ini menuju simpul akhir
f	g + h

3. Tambahkan simpul awal n ke open list
4. Ulangi langkah-langkah dibawah ini :
   a. Cari node dengan nilai f terendah pada open list lalu anggap sebagai node saat ini
   b. Ubah open list menjadi closed list
   c. Tiap node yang terjangkau oleh node saat ini :
      i. Jika terdapat pada closed list maka abaikan
      ii. Jika tidak terdapat pada open list, tambahkan pada open list dengan node parent nya adalah node saat ini. Hitung nilai g, h, dan f juga.
      iii. Jika sudah ada di open list, cek apakah node ini merupakan rute yang lebih baik. Jika iya, maka ubah node parentnya menjadi node saat ini dan hitung ulang nilai g dan f.
      iv. Loop berhenti ketika :
         - Menambahkan node akhir pada closed list
         - Gagal menemukan node akhir dan open list kosong
5. Telusuri balik path node akhir ke node awal, maka inilah rute terpendeknya.

IV. HASIL TESTING

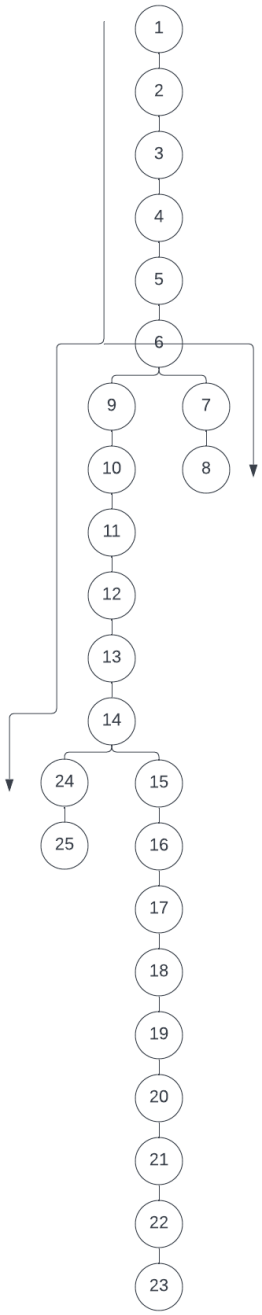
Setelah membuat pseudocode untuk masing-masing algoritma DFS, BFS, dan A*, langkah selanjutnya ialah testing masing-masing algoritma menggunakan *Maze Problem* yang sesungguhnya. Untuk Maze pertama ini sangat sederhana, dan akan dilihat berapa banyak langkah yang diperlukan untuk mencapai titik e dari titik s. Maze-nya sendiri sudah dikonversi dalam bentuk graf sehingga akan lebih mudah untuk pengujian dengan menggunakan algoritma DFS, BFS, dan A*.



Gambar 4. Labirin

A. BFS (Breadth First Search)

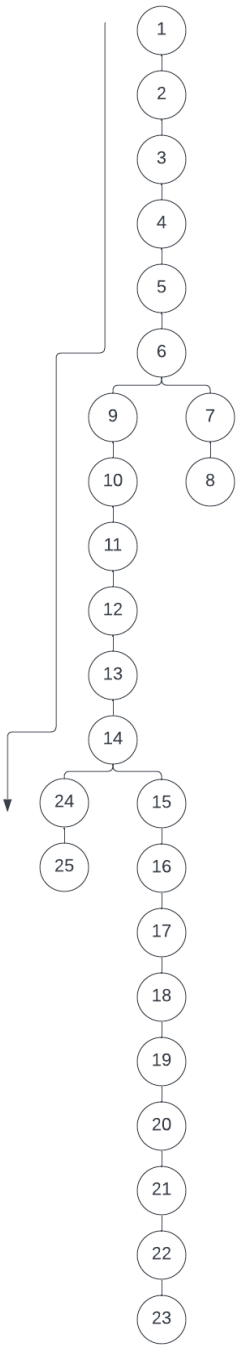
Pada algoritma BFS yaitu akan melakukan traversal secara meluas, tetapi karena maze yang sederhana, jadi BFS tidak akan terlihat terlalu berat untuk perhitungannya. Sebelum memasuki perhitungan, jika graf diatas diubah menjadi graf dalam bentuk tree, maka akan menjadi seperti berikut.



Gambar 5. Graf BFS

Dalam algoritma BFS, awal mula berawal dari node 1 hingga node 6 tiap node hanya akan masuk pada queue dan diambil lagi. Namun pada saat, node 6 dipanggil, akan membangkitkan 2 node child yaitu node 7 dan 9. Setelah itu, akan dipilih secara heuristik, karena node awal berada di kiri atas dan node akhir di kanan bawah, maka akan lebih memprioritaskan node yang kanan atau bawah terlebih dahulu sehingga akan mengambil node 9. Lalu setelah node 9 dan memasukkan node 10 pada queue, maka akan mengecek node 7 dan memasukkan node 8 pada queue. Selanjutnya akan dipanggil node 10 dan memasukkan node 11 pada queue. Lalu akan memanggil node 8 dan berhenti karena bukan node goal dan sudah buntu. Selanjutnya dari node 11 hingga node 14 akan hanya memasukkan 1 node pada queue dan mengambilnya. Pada node 14 akan memasukkan 2 node yakni 24 dan 15, karena node 24 berada di kanan (pada labirin) sehingga diprioritaskan, dan ternyata node 24 adalah node goal. Oleh karena itu pencarian berhenti sehingga rute pencariannya adalah 1-2-3-4-5-6-9-7-10-8-11-12-13-14-24 (15 langkah).

B. DFS (Depth First Search)



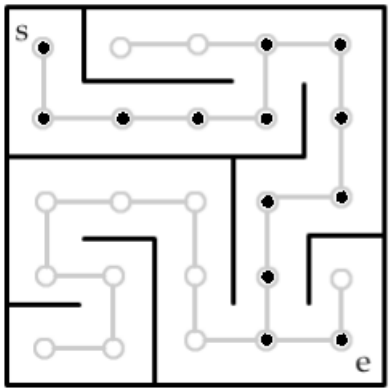
Gambar 6. Graf DFS

Pada algoritma DFS yaitu akan melakukan traversal secara mendalam, tetapi karena maze yang sederhana, jadi algoritma ini juga tidak akan terlihat terlalu berat untuk perhitungannya. Sebelum memasuki perhitungan, jika graf diatas diubah menjadi graf dalam bentuk tree, maka akan menjadi seperti Gambar 6.

Dalam algoritma DFS, awal mula berawal dari node 1 hingga node 6 tiap node hanya akan masuk pada queue dan diambil lagi. Namun pada saat, node 6 dipanggil, akan membangkitkan 2 node child yaitu node 7 dan 9. Setelah itu, akan dipilih secara heuristik, karena node awal berada di kiri atas dan node akhir di kanan bawah, maka akan lebih memprioritaskan node yang kanan atau bawah terlebih dahulu sehingga akan mengambil node 9. Lalu setelah node 9 diambil, maka akan langsung melakukan traversal ke kedalamannya sehingga dari node 9 hingga node 14 akan hanya membangkitkan 1 node saja. Lalu, pada node 14 akan membangkitkan 2 node yakni 24 dan 15, karena node 24 berada di kanan (pada labirin) sehingga diprioritaskan, dan ternyata node 24 adalah node goal. Oleh karena itu pencarian berhenti sehingga rute pencariannya adalah 1-2-3-4-5-6-9-10-11-12-13-14-24 (13 langkah).

*C. Algoritma A**

Sesuai dengan langkah algoritma A*, maka untuk rute pencariannya akan menjadi 1-2-3-4-5-6-9-10-11-12-13-14-24 (13 langkah). Pada penyelesaian labirin dengan algoritma A*, fungsi estimasi cost yang digunakan pada h adalah menghitung jarak node saat ini dengan node goal, yaitu jarak axis + ordinatnya.

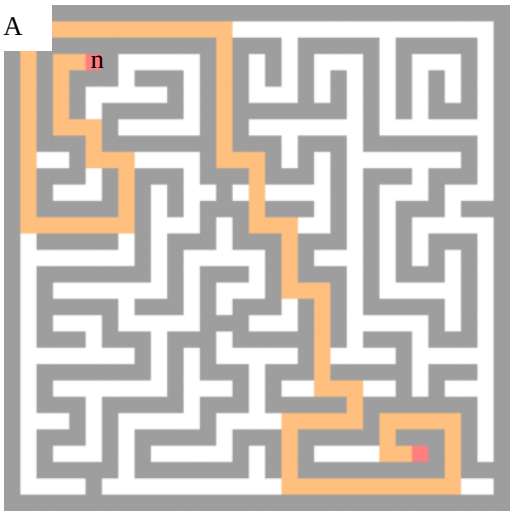


Gambar 7. Hasil Algoritma A*

Node saat ini	Node tetangga	g	h	f	Node yang dipilih
1	2	1	7	8	2
2	3	2	6	8	3
3	4	3	5	8	4
4	5	4	4	8	5
5	6	5	5	10	6
6	7	6	6	12	9
	9	6	4	10	
9	10	7	3	10	10
10	11	8	2	10	11
11	12	9	3	12	12
12	13	10	2	12	13
13	14	11	1	12	14
14	15	12	2	14	24
	24	13	0	13	

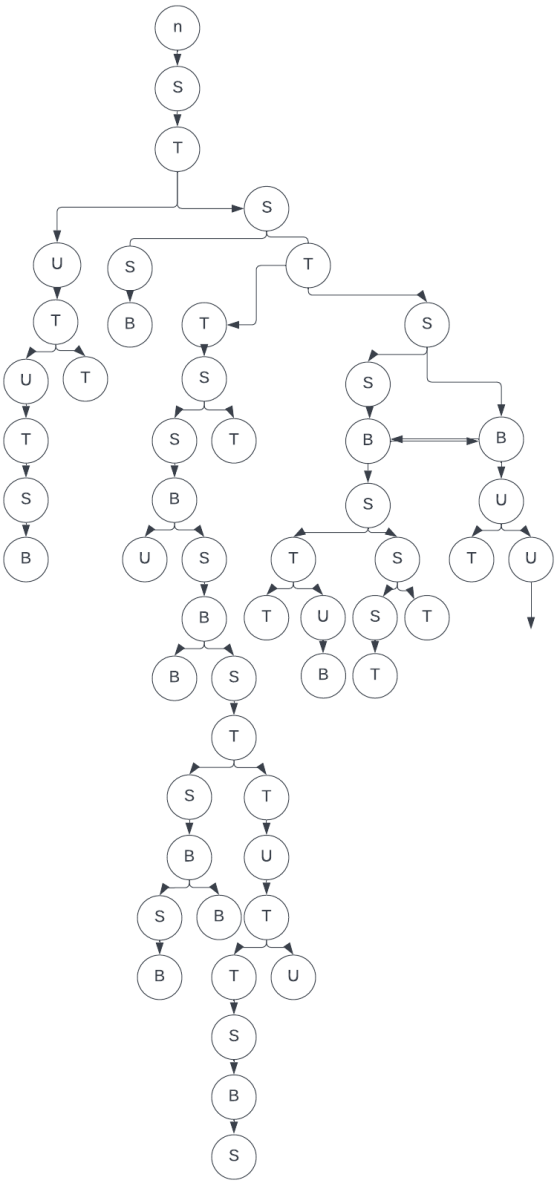
Namun, algoritma pencariannya memerlukan perhitungan yang lebih karena harus memperkirakan cost kedepannya. Untuk kasus Maze yang sederhana seperti ini memang terlihat lebih memakan banyak komputasi, tetapi untuk kasus Maze yang lebih rumit hal ini akan lebih efektif karena sangat mengurangi potensi untuk salah jalur sehingga mengurangi komputasinya.

Ketiga algoritma diatas telah diuji pada *Maze Problem* yang sederhana dan terlihat bahwa algoritma DFS lah yang paling membutuhkan komputasi yang lebih kecil, tetapi apa jadinya jika Maze-nya semakin rumit? akan dicoba untuk algoritma BFS dan DFS pada salah satu *Maze* yang lebih rumit pada Gambar 8 sehingga akan menghasilkan graf hingga rute sampai A dai n sebagaimana terlihat pada Gambar 9.



Gambar 8. Maze Rumit

Untuk persoalan labirin yang rumit akan jauh lebih memakan komputasi terlebih lagi bagi algoritma BFS yang melakukan traversal layer per layer. Untuk algoritma DFS sedikit lebih menguntungkan, walaupun untuk komputasinya tetap saja besar yakni eksponensial.



Gambar 9. Graf n hingga A

V. KESIMPULAN

Maze atau Labyrinth merupakan sebuah puzzle yang sudah umum di masa sekarang ini. Persoalan labirin dapat diselesaikan dengan algoritma BFS, DFS, maupun A*. Untuk algoritma BFS dan DFS, semakin rumit labirin yang akan diselesaikan maka semakin besar pula kompleksitas dari algoritma tersebut karena merupakan penyelesaian eksponensial. Sementara, algoritma A* melakukan estimasi cost sehingga untuk persoalan labirin yang lebih besar akan lebih optimal dan lebih sedikit dari komputasinya.

A. BFS (Breadth First Search)

Hasil eksperimen untuk BFS menunjukkan bahwa BFS menempati posisi komputasi paling lama yakni 15 langkah karena worst case dari BFS adalah ketika goal node berada di kedalaman sangat besar. Sehingga untuk average persoalan maze yang memang ditujukan untuk lintasan yang besar atau kedalaman graf yang besar, maka algoritma ini kurang cocok jika digunakan pada penyelesaian persoalan labirin.

B. DFS (Depth First Search)

Pada algoritma DFS, terlihat bahwa langkah yang diperlukan untuk mencapai goal node adalah paling cepat yakni hanya 13 langkah saja. Hal ini dikarenakan memang persoalan labirin yang diberikan merupakan persoalan labirin yang sederhana. Namun, jika dibandingkan dengan algoritma BFS, algoritma ini lebih cocok digunakan pada persoalan labirin karena rata-rata persoalan labirin mengambil node yang paling jauh, sehingga cocok dengan algoritma DFS. Walaupun pada persoalan labirin yang rumit tetap saja DFS memiliki kelemahan jika dibandingkan algoritma A*.

*C. Algoritma A**

Untuk algoritma A* juga menghasilkan rute 13 langkah, yakni dengan fungsi estimasi cost yang tepat. Kompleksitas algoritma ini sangat tergantung pada fungsi estimasi cost yang merupakan fungsi heuristik yang digunakan untuk meminimalkan komputasi seperti pada DFS dan BFS. Untuk pemilihan fungsi estimasi cost yang tepat tentu akan menghasilkan algoritma yang super cepat pula karena tidak akan seperti algoritma yang berkompleksitas eksponensial tetapi mungkin hanya linear.

VIDEO LINK AT YOUTUBE

https://youtu.be/VeYlsKwdsTA

REFERENCES

[1] C. F. Vara, “Labyrinth and Maze: Video Game Navigation Challenges”, Academia, 2007. Tersedia : https://www.academia.edu/284138/_Labyrinth_and_Maze_Video_Game_Navigation_Challenges_?auto=citations&from=cover_page [23 Mei 2022]

[2] N. H. Barnouti, dkk, “Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm”, Journal of Computer and Communications, 2016, 4, 15-25. Tersedia : https://www.scirp.org/html/2-1730422_70460.htm?pagespeed=noscript [23 Mei 2022]

[3] Kapitan, D.Y., Rybin, A.E., Vasiliev, E.V., Perzhu, A.V. and Andriuschenko, P.D., 2019. The Comparison of DFS and BFS Methods on 2D Ising Model. In CEUR Workshop Proceedings (Vol. 2426, pp. 147-152). Tersedia : http://ceur-ws.org/Vol-2426/paper22.pdf [23 Mei 2022]

[4] Bahan Kuliah IF2211 Strategi Algoritmik oleh Rinaldi Munir & Nur Ulfa Maulidevi

[5] Ansari, A., Sayyed, M.A., Ratlamwala, K. and Shaikh, P., 2015. An optimized hybrid approach for path finding. arXiv preprint arXiv:1504.02281. Tersedia : https://arxiv.org/abs/1504.02281 [23 Mei 2022]

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, 20 Mei 2022

Roby Purnomo
13520106