Penerapan Algoritma *Depth First Search (Dfs)* Dalam *Procedural Generating Maze* Pada *Game Mystic Maze*

Rizqy Nur Mauliddinah[1], Zainal Abidin[2], Lingga Andrian Maulana[3], Fresy Nugroho[4], Ahmad Fahmi Karami[5]

[1,2,3,4,5]Universitas Islam Negeri Maulana Malik Ibrahim Malang

*e-mail: 220605110024@student.uin-malang.ac.id

Abstrak − *Mystic Maze* merupakan salah satu jenis permainan yang mengasah logika dan strategi, di mana pemain harus menemukan jalur dari titik awal hingga mencapai tujuan yang ditentukan. Namun, menciptakan maze yang menantang, unik, dan efisien secara dinamis masih menjadi tantangan dalam pengembangan *game modern*. Penelitian ini bertujuan untuk mengembangkan game maze berbasis procedural generation menggunakan algoritma *Depth First Search (DFS)* yang diimplementasikan pada Unity. Metode DFS dipilih karena kemampuannya untuk menjelajahi jalur secara mendalam dan membentuk jalur solusi tunggal. Selanjutnya, penerapan teknik *shuffling* pada DFS digunakan untuk memastikan maze yang dihasilkan selalu unik pada setiap sesi permainan. Penelitian ini dimulai dengan inisialisasi seluruh area *maze* sebagai dinding, kemudian jalur dibentuk secara bertahap menggunakan DFS dengan memanfaatkan struktur data *stack* untuk mencatat jalur eksplorasi. Hasil implementasi menunjukkan bahwa maze dengan dimensi 81x81 node dapat dihasilkan secara efisien dalam waktu singkat, dilengkapi fitur area tengah kosong berukuran 8x8 untuk mempermudah pemain memulai permainan, serta pintu keluar atau *finish game* yang jelas. Penelitian ini menunjukkan potensi untuk diterapkan tidak hanya sebagai alat hiburan tetapi juga dalam edukasi, khususnya untuk melatih logika dan pemecahan masalah. Namun, penelitian ini memiliki keterbatasan pada variasi jalur solusi dan pengaturan tingkat kesulitan yang memerlukan pengembangan algoritma tambahan di masa depan.

Kata kunci: *Procedural Generating Maze, Depth First Search, Game Mystic Maze.*

PENDAHULUAN

Perkembangan teknologi dan minat masyarakat terhadap game terus mengalami peningkatan signifikan (Syarif et al., 2022). Industri game telah menjadi salah satu sektor hiburan yang berkembang pesat, baik dalam aspek teknologi maupun kreativitas. Dengan kemajuan teknologi perangkat keras dan lunak, pengembangan game kini semakin kompleks, seperti penggunaan engine Unity yang mendukung berbagai jenis game berbasis mobile maupun desktop, baik 2D maupun 3D (Devi Novayanti & Puritan Wijaya ADH, 2023). Game tidak hanya menjadi alat hiburan tetapi juga mampu melatih pola pikir dan keterampilan pemecahan masalah pemain (Yulyanto et al., 2023). Pesatnya perkembangan industri game didukung oleh ketersediaan perangkat keras dan lunak yang semakin terjangkau, membuat game tidak hanya populer di kalangan anak muda tetapi

juga lintas usia (Muchsin Sanin et al., 2024). Selain menjadi alat untuk hiburan, *game* juga bisa dijadikan sebagai sarana pendidikan bagi anak (Mayer, 2019). Dengan menggunakan *game* sebagai sarana pendidikan, terutama untuk belajar, maka akan lebih efektif karena dapat meningkatkan daya tarik anak dan juga mengubah *mindset* anak ketika akan belajar sehingga dapat menciptakan suasana belajar yang lebih menyenangkan (Junaidi, 2024).

Salah satu konsep permainan yang menarik perhatian adalah labirin (*maze*), yang memadukan elemen logika dan strategi dalam pencariannya (Faizah et al., 2023). Maze merupakan puzzle berbentuk jalur bercabang, di mana pemain harus menemukan jalan dari titik awal ke tujuan yang ditentukan. Permainan ini memiliki sejarah panjang, berasal dari struktur labirin kuno seperti yang dirancang Daedalus untuk Raja Minos (Pribadi & Kom, 2015). Dalam konteks modern, maze banyak diadaptasi ke dalam bentuk permainan digital untuk mengasah kemampuan logika dan kecerdasan visual pemain (Taufiq Subagio & Martha, 2015).

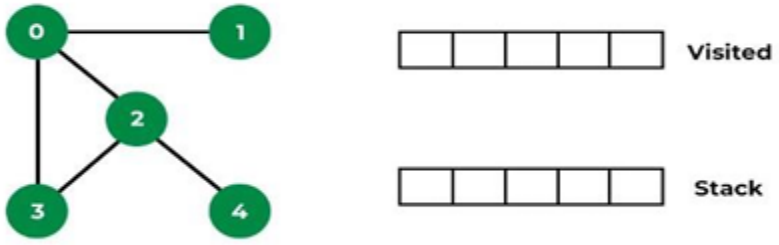
Mekanisme penyelesaian maze sering kali melibatkan pendekatan algoritmik seperti Depth First Search (DFS), yang bekerja dengan menjelajahi jalur hingga menemukan solusi atau menemui jalan buntu, di mana proses backtracking dilakukan untuk mencari jalur lain (Wang et al., 2021). DFS sangat cocok digunakan dalam permainan labirin yang memiliki satu jalur solusi karena algoritma ini memungkinkan eksplorasi yang mendalam dan efisien (Kumar et al., 2019). Meskipun demikian, DFS memiliki keterbatasan dalam memberikan solusi optimal karena tidak semua ruang pencarian dapat terinspeksi sepenuhnya (Purwandari, 2010).

Melihat pentingnya perancangan maze yang menantang namun tetap efisien, penelitian ini mengusulkan penerapan algoritma DFS untuk membangun maze secara procedural dalam game "Mystic Maze". Dengan pendekatan ini, maze akan dihasilkan secara dinamis, memungkinkan pengalaman bermain yang unik dan tidak repetitif bagi pemain. Penelitian ini bertujuan untuk mengeksplorasi bagaimana algoritma DFS dapat diintegrasikan dengan teknologi Unity untuk menciptakan game berbasis maze yang interaktif dan inovatif (Gajewski et al., 2022). Di samping itu, penelitian ini juga diharapkan memberikan wawasan lebih mendalam tentang implementasi algoritma DFS dalam pengembangan game modern.

METODE

Penelitian ini menggunakan algoritma *Depth First Search (DFS)* dalam pembuatan *maze. Depth first search (DFS)* adalah algoritma umum traversal atau pencarian pada pohon, struktur pohon, atau graf (Tan et al., 2021). DFS adalah pencarian yang berjalan dengan meluaskan anak akar pertama dari pohon pencarian yang dipilih dan berjalan dalam dan lebih dalam lagi sampai simpul tujuan ditemukan, atau sampai menemukan simpul yang tidak punya anak (Iqbal et al., 2022). Kemudian, pencarian backtracking, akan kembali ke simpul yang belum selesai ditelusuri (Refan Rahmat Fauzi et al., 2023). Dalam implementasi nonrekursif , semua simpul yang diluaskan ditambahkan ke LIFO stack untuk penelusuran. Berikut ini langkah logika penerapan algoritma DFS (Inggiantowi, 2009).

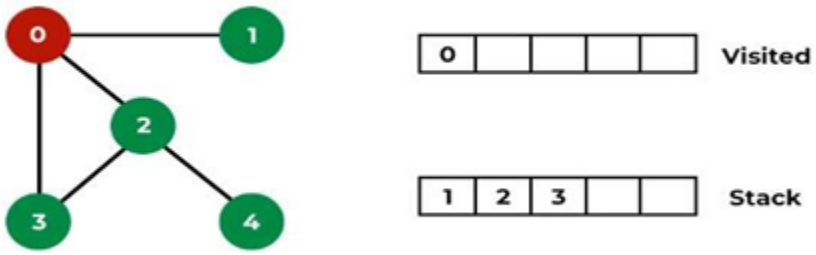
- Pada langkah pertama algoritma Depth First Search (DFS), proses dimulai dengan inisialisasi terhadap dua elemen utama, yaitu tabel Visited dan Stack, yang keduanya masih dalam kondisi kosong. Tabel Visited digunakan untuk mencatat node-node yang telah dikunjungi selama eksplorasi berlangsung, sementara Stack berfungsi sebagai struktur data yang melacak jalur yang sedang dieksplorasi. Pada tahap awal ini, belum ada node yang diproses, sehingga tabel Visited dan Stack kosong. Graph yang digunakan dalam proses ini terdiri dari lima node (0, 1, 2, 3, dan 4) yang terhubung melalui edge, mencerminkan hubungan antara node dalam maze atau puzzle. Langkah awal ini menggambarkan kondisi awal sebelum DFS mulai bekerja, di mana nantinya node awal akan dimasukkan ke dalam stack sebagai permulaan eksplorasi .



Gambar 1. Langkah 1 Logika Penerapan DFS

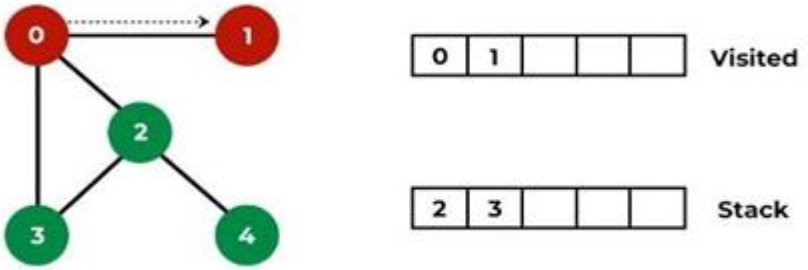
- Pada langkah kedua algoritma Depth First Search (DFS), proses eksplorasi dimulai dengan memilih node 0 sebagai titik awal. Node 0 langsung ditandai sebagai telah dikunjungi, sehingga node ini dimasukkan ke dalam tabel Visited. Tabel Visited kini

mencatat bahwa node 0 telah dieksplorasi. Selanjutnya, semua tetangga langsung dari node 0, yaitu node-node yang terhubung dengan edge ke node 0, dimasukkan ke dalam Stack untuk diproses lebih lanjut. Berdasarkan graph yang diberikan, node 0 memiliki tiga tetangga langsung, yaitu node 1, node 2, dan node 3. Oleh karena itu, ketiga node ini dimasukkan ke dalam Stack secara berurutan (Pardede et al., 2022).



Gambar 2. Langkah 2 Logika Penerapan DFS

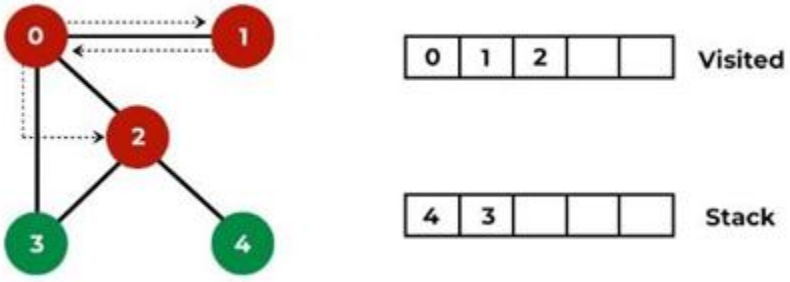
- Pada langkah ketiga algoritma Depth First Search (DFS), proses berlanjut dengan mengambil node 1 dari Stack untuk dieksplorasi. Setelah diambil, node 1 ditandai sebagai telah dikunjungi dengan cara memasukkannya ke dalam tabel Visited. Sekarang, tabel Visited mencatat bahwa node 1 telah dieksplorasi, sehingga berisi node 0 dan 1. Setelah itu, algoritma memeriksa tetangga langsung dari node 1. Berdasarkan graph, satu-satunya tetangga langsung dari node 1 adalah node 0, tetapi node ini sudah terdaftar dalam tabel Visited, sehingga node 0 tidak dimasukkan lagi ke dalam Stack. Oleh karena itu, tidak ada perubahan pada isi Stack setelah langkah ini.



Gambar 3. Langkah 3 Logika Penerapan DFS

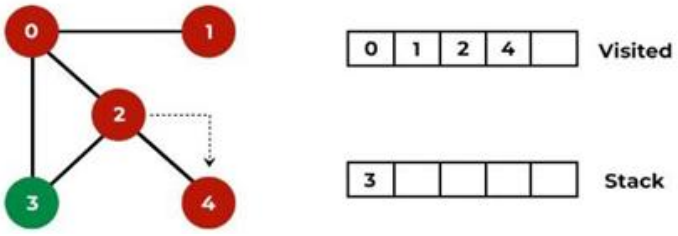
- Setelah selesai menelusuri node 1, algoritma Depth First Search (DFS) melakukan backtracking ke node 0, karena node 1 adalah simpul ujung (tidak memiliki tetangga baru yang belum dikunjungi). Selanjutnya, DFS melanjutkan eksplorasi dengan memilih node 2 dari Stack untuk diproses. Node 2 kemudian ditandai sebagai telah dikunjungi dengan memasukkannya ke dalam tabel Visited. Kini, tabel Visited

mencatat 0, 1, dan 2 sebagai node yang telah dieksplorasi. Setelah itu, algoritma memeriksa tetangga langsung dari node 2. Berdasarkan graph, node 2 memiliki dua tetangga, yaitu node 3 dan node 4. Dari kedua tetangga tersebut, node 3 sudah berada di dalam Stack dari langkah sebelumnya, sehingga hanya node 4 yang perlu ditambahkan ke Stack pada tahap ini.



Gambar 4. Langkah 4 Logika Penerapan DFS

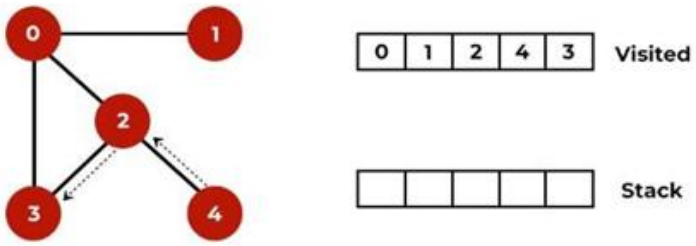
- Pada langkah ini, algoritma Depth First Search (DFS) melanjutkan eksplorasi menuju node 4, yang merupakan node terdalam yang tersedia dalam Stack. Node 4 diambil dari Stack untuk diproses dan langsung ditandai sebagai telah dikunjungi dengan memasukkannya ke dalam tabel Visited. Dengan demikian, tabel Visited kini mencatat bahwa node 0, 1, 2, dan 4 telah dieksplorasi. Setelah ditelusuri, algoritma memeriksa tetangga langsung dari node 4. Berdasarkan graph, satu-satunya tetangga langsung dari node 4 adalah node 2. Namun, node 2 telah tercatat dalam tabel Visited sebelumnya, sehingga node 4 tidak memiliki tetangga baru untuk dieksplorasi lebih lanjut. Oleh karena itu, Stack kini hanya berisi node 3, yang akan dieksplorasi pada langkah berikutnya.



Gambar 5. Langkah 5 Logika Penerapan DFS

- Pada langkah terakhir algoritma Depth First Search (DFS), karena node 4 adalah simpul ujung tanpa tetangga baru yang belum dikunjungi, algoritma melakukan backtracking kembali ke node 2. Dari node 2, algoritma melanjutkan eksplorasi ke

node 3, yang merupakan node terakhir dalam Stack. Node 3 diambil dari Stack, ditandai sebagai telah dikunjungi, dan dimasukkan ke dalam tabel Visited. Kini tabel Visited mencatat seluruh node dalam graph (0, 1, 2, 4, dan 3) telah ditelusuri. Setelah eksplorasi node 3, algoritma memeriksa tetangga langsungnya. Berdasarkan graph, satu-satunya tetangga node 3 adalah node 2, yang sudah tercatat dalam tabel Visited sebelumnya. Karena tidak ada lagi tetangga baru yang belum dikunjungi, Stack menjadi kosong, menandakan bahwa semua node dalam graph telah dieksplorasi.



Gambar 6. Langkah 6 Logika Penerapan DFS

Proses ini mengakhiri algoritma DFS untuk graph yang diberikan. Dengan mekanisme *backtracking* dan eksplorasi mendalam hingga ke node terakhir, DFS berhasil mengunjungi seluruh node dalam graph, menunjukkan efisiensinya dalam mengeksplorasi struktur graph secara sistematis (Afif Nawawi et al., 2023). Seluruh proses tersebut

```
DFS(Graph G, Node start):
   stack ← empty stack
   visited ← empty list

   Push start to stack

   While stack is not empty:
      current ← stack.Pop()

      If current is not in visited:
         Add current to visited

         For each neighbor in Neighbors(current):
            If neighbor is not in visited:
               Push neighbor to stack

   Return visited
```

Gambar 7. Pseudocode Algoritma DFS

nantinya akan diterapkan pada *script* logika maze yang akan dibuat. Berikut ini adalah pseudocode penerapan algoritma DFS pada *maze*.

HASIL KARYA UTAMA DAN PEMBAHASAN

Penelitian ini menghasilkan game maze yang dirancang secara procedural menggunakan algoritma Depth First Search (DFS) dengan implementasi pada Unity. Maze yang dihasilkan memiliki dimensi 81x81 node dengan area kosong di pusat berukuran 8x8, yang dirancang untuk mempermudah pemain memulai permainan di tengah labirin. Proses generasi maze dimulai dengan menginisialisasi seluruh area sebagai dinding, kemudian jalur dibentuk menggunakan logika DFS. Algoritma ini memanfaatkan stack untuk mencatat jalur eksplorasi, memastikan setiap node dieksplorasi secara mendalam sebelum kembali ke jalur lain melalui proses backtracking. Implementasi kode dari algoritma DFS yang digunakan disajikan pada Gambar 8. Selain itu, untuk memastikan maze yang dihasilkan tidak repetitif, setiap arah gerakan diacak menggunakan fungsi *shuffling* sebelum eksplorasi, dimana kodenya terdapat pada gambar 9.

```
// Algoritma Depth-First Search untuk membuat maze
void GenerateMazeDFS()
{
    Stack<Vector2Int> stack = new Stack<Vector2Int>(); // Tumpukan untuk menyimpan posisi
    Vector2Int current = new Vector2Int(1, 1); // Posisi awal (harus ganjil)
    map[current.x, current.y] = 0; // Tandai sebagai jalan
    stack.Push(current);

    // Arah pergerakan (kanan, kiri, atas, bawah)
    Vector2Int[] directions = { new Vector2Int(2, 0), new Vector2Int(-2, 0), new Vector2Int(0, 2), new Vector2Int(0, -2) };

    while (stack.Count > 0)
    {
        current = stack.Pop();
        ShuffleArray(directions); // Acak arah untuk membuat maze lebih acak

        foreach (var dir in directions)
        {
            Vector2Int neighbor = current + dir; // Tetangga yang dicek

            if (IsInBounds(neighbor) && map[neighbor.x, neighbor.y] == 1)
            {
                // Kosongkan dinding antara posisi saat ini dan tetangga
                Vector2Int wall = current + dir / 2;
                map[wall.x, wall.y] = 0;
                map[neighbor.x, neighbor.y] = 0;

                // Tambahkan tetangga ke stack untuk dieksplorasi
                stack.Push(neighbor);
            }
        }
    }
```

Gambar 8. Kode untuk Membuat Maze dengan algoritma DFS

```
// Mengacak array (untuk arah gerakan acak)
void ShuffleArray(Vector2Int[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        int rnd = Random.Range(0, array.Length);
        Vector2Int temp = array[i];
        array[i] = array[rnd];
        array[rnd] = temp;
    }
}
```

Gambar 9. Kode untuk Penerapan Teknik Shuffling pada Algoritma DFS

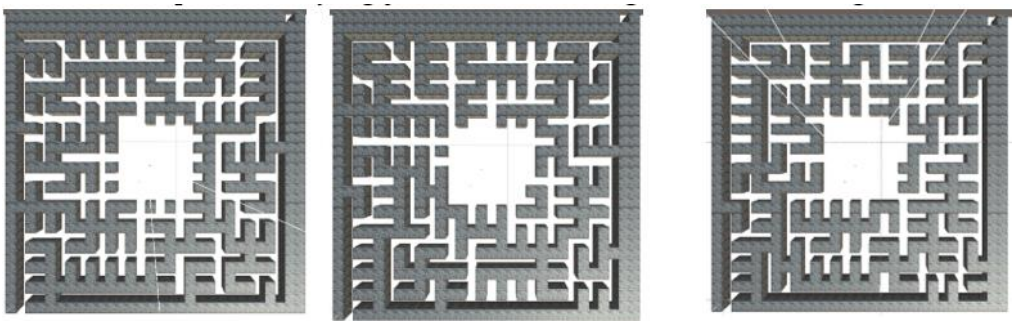
```
// Kosongkan area tengah berbentuk 8x8
void EnsureCenterArea()
{
    int centerX = width / 2;
    int centerZ = depth / 2;

    int areaSize = 8; // Ukuran area kosong (8x8)
    int halfSize = areaSize / 2;

    // Kosongkan area 8x8 di tengah
    for (int z = centerZ - halfSize; z < centerZ + halfSize; z++)
    {
        for (int x = centerX - halfSize; x < centerX + halfSize; x++)
        {
            map[x, z] = 0; // Kosongkan dinding
        }
    }
}
```

Gambar 10. Kode untuk Membuat Room di Tengah Maze berukuran 8 x 8

Hasil pembuatan maze ini juga memiliki beberapa fitur utama. Pertama, area tengah maze dikosongkan dengan ukuran **8x8** untuk memberikan ruang aman bagi pemain memulai permainan, dimana kodenya terdapat pada **gambar 10**. Kedua, maze dilengkapi dengan pintu masuk di sudut kiri atas dan pintu keluar di sudut kanan bawah, memberikan mekanisme permainan yang jelas**.** Selain itu, algoritma DFS menghasilkan maze yang bercabang dan menantang dengan jalur solusi tunggal, yang sangat cocok untuk game berbasis logika dan strategi. Gambar 11 menampilkan beberapa hasil percobaan pembangkitan maze menggunakan algoritma DFS. Maze yang dihasilkan menunjukkan struktur bercabang dengan jalur yang menantang, sesuai dengan karakteristik permainan puzzle.



(a) **(b)** **(c)**

Gambar 11(a). Hasil Percobaan 1 dari Maze yang telah dibuat, **(b).** Hasil Percobaan 2 dari Maze yang telah dibuat, **(c).** Hasil Percobaan 3 dari Maze yang telah dibuat

Proses pengembangan maze dengan DFS pada Unity menunjukkan efisiensi dalam menghasilkan maze berukuran besar dalam waktu singkat. Selain itu, pengacakan arah gerakan menggunakan teknik *shuffling* memungkinkan maze yang dihasilkan selalu unik di setiap sesi permainan (Academy et al., 2017). Namun, karya ini memiliki keterbatasan, yaitu hanya menghasilkan satu jalur solusi, yang dapat membatasi variasi eksplorasi pemain. Penyesuaian tingkat kesulitan maze juga membutuhkan algoritma tambahan, karena DFS tidak secara langsung mempertimbangkan aspek ini.

Keunggulan dari karya ini meliputi efisiensi generasi maze, fleksibilitas ukuran maze, serta kemudahan implementasi melalui Unity. Di sisi lain, untuk pengembangan lebih lanjut, algoritma tambahan seperti pembobotan jalur atau integrasi DFS dengan A* dapat diterapkan untuk menciptakan jalur yang lebih kompleks. Maze ini tidak hanya terbatas bisa diterapkan pada *game Mystic Maze.* Namun, juga memiliki potensi untuk digunakan dalam konteks edukasi, seperti melatih logika dan pemecahan masalah pada anak-anak, sehingga memberikan manfaat strategis bagi sektor pendidikan maupun hiburan.

KESIMPULAN

Penelitian ini berhasil mencapai tujuan utama, yaitu mengembangkan game maze berbasis procedural generation menggunakan algoritma Depth First Search (DFS) dengan implementasi pada Unity. Maze yang dihasilkan memiliki dimensi 81x81 node dengan fitur area tengah kosong berukuran 8x8 untuk memberikan ruang aman bagi pemain memulai permainan. Selain itu, maze dilengkapi dengan pintu keluar atau *finish* yang jelas, sehingga memberikan pengalaman bermain yang terstruktur. Proses generasi maze dilakukan dengan efisien menggunakan DFS, yang memanfaatkan struktur data stack

untuk mencatat jalur eksplorasi. Teknik *shuffling* yang diterapkan memastikan maze yang dihasilkan selalu unik di setiap sesi permainan, memberikan tantangan yang menarik bagi pemain.

Hasil penelitian ini menunjukkan bahwa algoritma DFS dapat diterapkan secara efektif untuk membentuk maze bercabang dan menantang, meskipun memiliki keterbatasan pada variasi jalur solusi dan pengaturan tingkat kesulitan. Penelitian ini memberikan kontribusi signifikan dalam pengembangan teknologi game, terutama dalam konteks generasi procedural yang efisien dan fleksibel. Selain itu, karya ini juga memiliki potensi untuk diterapkan dalam bidang edukasi, khususnya untuk melatih logika dan kemampuan pemecahan masalah, yang relevan bagi anak-anak maupun pelajar.

Penelitian ini juga membuka peluang untuk pengembangan lebih lanjut, seperti integrasi dengan algoritma A* untuk meningkatkan kompleksitas jalur atau pembobotan tingkat kesulitan untuk memenuhi kebutuhan pengguna yang lebih beragam. Dengan demikian, hasil penelitian ini tidak hanya memberikan solusi inovatif dalam pengembangan game berbasis maze, tetapi juga berkontribusi dalam pengembangan ilmu komputer, khususnya dalam penerapan algoritma pada sistem procedural generation, serta memiliki manfaat strategis bagi masyarakat dalam mendukung edukasi dan hiburan yang berbasis logika dan strategi.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada dosen pengampu mata kuliah Pemrograman Game atas bimbingan, arahan, dan dukungan yang diberikan selama proses penulisan jurnal ini. Ucapan terima kasih juga disampaikan kepada teman-teman kelompok atas kerja sama, dukungan moril, dan kontribusi yang tak ternilai dalam menyelesaikan penelitian ini. Semoga hasil dari penelitian ini dapat bermanfaat bagi pengembangan ilmu pengetahuan di bidang pemrograman game.

DAFTAR PUSTAKA

Academy, B., Bontchev, B., & Panayotova, R. (2017). Towards Automatic Generation Of Serious Maze Games For Education. *ACM Computing Classification System* (Vol. 11, Issue 4).

Afif Nawawi, A., Zakiyah, Y., & Rosyani, P. (2023). Analisa Penggunaan Metode Dfs (Depth First Search) Di Sistem Pakar Sebuah Penyakit: Systematic Literature Review. *Jurnal Artificial Inteligent dan Sistem Penunjang Keputusan* (Vol. 1, Issue 1).

Devi Novayanti, P., & Puritan Wijaya ADH, I. (2023). Identifikasi Software Dan Metode Pengembangan Game Di Indonesia Menggunakan Systematic Literature Review. *Jurnal Teknik Informatika Dan Sistem Informasi*, *10*(4). http://jurnal.mdp.ac.id

Faizah, N., Ainol, A., & Kiromi, I. H. (2023). Implementation Of Maze Games In Learning For Children's Cognitive Development At Ra Al-Khairat. *Golden Age : Jurnal Pendidikan Anak Usia Dini*, *7*(1), 17–26. https://doi.org/10.29313/ga:jpaud.v7i1.11640

Gajewski, S., El Mawas, N., & Heutte, J. (2022). A Systematic Literature Review of Game Design Tools. *International Conference on Computer Supported Education, CSEDU - Proceedings*, *2*, 404–414. https://doi.org/10.5220/0011137800003182

Inggiantowi, H. (2009). *Perbandingan Algoritma Penelusuran Depth First Search dan Breadth First Search pada Graf serta Aplikasinya*.

Iqbal, M., Simanungkalit, A., Usman, A., & Budiman, A. (2022). *Attribution-ShareAlike 4.0 International Some rights reserved Recursive Algorithm Implementasi Depth First Search dalam Game Tic Tac Toe Berbasis Android*.

Junaidi, J. (2024). *Aplikasi Game Maze Untuk Meningkatkan Semangat Anak Usia Dini Menggunakan Metode Game Development Life Cycle (GDLC)* (Vol. 4, Issue 2).

Kumar, N., Kaur, S., & Tech Student, M. (2019). A Review of Various Maze Solving Algorithms Based on Graph Theory. In *IJSRD-International Journal for Scientific Research & Development*| (Vol. 6). www.ijsrd.com

Mayer, R. E. (2019). Computer Games in Education. *Annual Review of Psychology Downloaded from Www.Annualreviews.Org. Guest (Guest)*. https://doi.org/10.1146/annurev-psych-010418

Muchsin Sanin, N., Sulthon, M. K., Aziz Aini, F., Dawud, M., Reza Ubaidillah, F., Fahmi Karami, A., & Nugroho, F. (2024). Implementasi FSM (Finite State Machine) Pada Game Muslims Explorer's Faithful Maze Adventure. In *JASTEN Jurnal Aplikasi Sains Teknologi Nasional* (Vol. 05).

Pardede, S. L., Athallah, F. R., Huda, Y. N., & Zain, F. D. (2022). A Review of Pathfinding in Game Development. *[CEPAT] Journal of Computer Engineering: Progress, Application and Technology*, *1*(01), 47. https://doi.org/10.25124/cepat.v1i01.4863

Pribadi, O., & Kom, S. (2015). Maze Generator dengan Menggunakan Algoritma Depth-First-Search. In *Jurnal TIMES: Vol. IV*.

Purwandari, N. (2010). *Games Applications 3D Maze To Know The Object Of Tourism In Indonesia Using Mobile*. http://www.gunadarma.ac.id

Refan Rahmat Fauzi, Mochammad Taufik Faturrohman, & Raihan Samhari. (2023). Penggunaan Algoritma Backtracking Pada Permainan Knight's Tour Dengan

Membandingkan Algoritma BFS Dan DFS. *Jurnal Ilmiah Teknik Informatika Dan Komunikasi*, *3*(2), 168–173. https://doi.org/10.55606/juitik.v3i2.512

Syarif, S., Hasanuddin, T., & Hasnawi, M. (2022). *Buletin Sistem Informasi dan Teknologi Islam INFORMASI ARTIKEL ABSTRAK*. *3*(1), 34–41.

Tan, C. S., Mohd-Mokhtar, R., & Arshad, M. R. (2021). A Comprehensive Review of Coverage Path Planning in Robotics Using Classical and Heuristic Algorithms. In *IEEE Access* (Vol. 9, pp. 119310–119342). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/ACCESS.2021.3108177

Taufiq Subagio, R., & Martha, D. (2015). Aplikasi Game Edukasi Labyrinth Wall 3d Menggunakan Unity 3D. In *JURNAL DIGIT* (Vol. 5, Issue 1).

Wang, M., Xu, X., Yue, Q., & Wang, Y. (2021). *A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search*. http://arxiv.org/abs/2101.12631

Yulyanto, A. B., Maudia Arsyad, F., & Sugiharto, T. (2023). Pengembangan Game Puzzle Find Grass Menggunakan. *Bulletin of Information Technology (BIT)*, *4*(2), 275–280. https://doi.org/10.47065/bit.v3i1