ISSN 2355-3286

Algoritma Fisher-Yates Shuffle dan Flood Fill sebagai Maze Generator pada Game Labirin

D. J. Owen Hoetama[1], Farica Perdana Putri[2], P.M. Winarno[3]
Program Studi Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
dominicus.jovian@student.umn.ac.id
farica@umn.ac.id
pmwinarno@umn.ac.id

Diterima 1 November 2018
Disetujui 21 Desember 2018

***Abstract*— Maze game is an interesting game and used to spend time. However, in the maze game, the level used for this game still uses static levels. Static levels make the maze shape stay the same if we play the same level. Thus, players will quickly feel bored because it finds the same complexity. Maze generator is a static level problem solution on the maze game. This research uses Fisher-Yates Shuffle algorithm and Flood Fill algorithm to make maze generator. Fisher-Yates Shuffle algorithm is used for wall position randomization and Flood Fill algorithm to keep the maze results to remain resolved. The results of the application implementation yielded 30 mazes and were tested using the Hamming Distance algorithm, yielding that the result of the maze formed is always different. The average percentage rate difference produced 48% each time the maze was formed. The results of the maze that was formed performed perfect maze checking with the result of 83.33% percentage.**

***Index Terms*— Fisher-Yates Shuffle, Flood Fill, Maze Generator, Hamming Distance**

I. PENDAHULUAN

Permainan Labirin merupakan permainan yang cukup menarik perhatian dan digunakan untuk membuang waktu luang [1]. Permainan Labirin adalah permainan yang bertujuan untuk mencari lokasi tujuan dari tempat semula [1]. *Static level* ditemukan dalam permainan labirin. *Level* yang sudah pernah dimainkan dalam permainan labirin akan selalu sama jika dimainkan kembali. Akibatnya pemain yang memainkan *level* yang sama akan menemukan kerumitan yang sama sehingga membuat pemain merasa bosan [2].

Random generator merupakan salah satu solusi untuk mengatasi *static level* [3]. Dalam permainan labirin, *random generator* diterapkan pada *maze generator* agar labirin yang dibentuk selalu berbeda setiap kali dimainkan [4].

Algoritma Fisher-Yates Shuffle dapat dijadikan referensi untuk diterapkan dalam sebuah aplikasi yang menggunakan metode pengacakan. Algoritma Fisher-Yates Shuffle merupakan cara yang optimal dengan waktu eksekusi yang efisien, serta dengan ruang penyimpanan memori yang tidak terlalu besar [5].

Algoritma Flood Fill merupakan metode yang umum digunakan untuk menyelesaikan permainan labirin dalam bentuk dinding [6]. Algoritma Flood Fill adalah algoritma yang menentukan area yang terhubung terhadap *node* pada *multidimensional array*. Sehingga, dapat ditentukan apakah setiap *node* saling berhubungan dan dapat dilalui [7] Algoritma Flood Fill mampu menangani segala macam bentuk susunan labirin secara efisien baik itu *perfect maze* ataupun *imperfect maze* karena langsung mengarah ke seluruh sel tujuan labirin [8].

Untuk mengatasi masalah *static level* pada permainan labirin, digunakan algoritma Fisher-Yates Shuffle sebagai pengacakan dinding untuk membuat *maze generator*. Namun, algoritma Fisher-Yates Shuffle hanya sebatas melakukan pengacakan, tidak dapat memastikan bahwa labirin dapat diselesaikan. Ditambahkan algoritma Flood Fill untuk menjaga hasil pengacakan algoritma Fisher-Yates Shuffle agar labirin yang terbentuk tetap dapat mencapai *finish*.

Tujuan dari penelitian ini adalah untuk mengimplementasikan algoritma Fisher-Yates Shuffle dan algoritma Flood Fill sebagai *maze generator* pada permainan labirin dengan tiga tingkat kesulitan (*easy*, *normal*, dan *hard*) yang setiap kali dimainkan, bentuk labirinnya selalu berubah.

II. TELAAH LITERATUR

A. Game Labirin 2D

Permainan Labirin adalah suatu jenis permainan yang terlihat sederhana namun mempunyai banyak teka-teki logika untuk menyelesaikannya. Tampilan dua dimensi (panjang dan lebar), hanya dapat dilihat dari satu sudut perspektif. Aplikasi permainan labirin 2D ini bersifat *single-user*. Tingkatan atau level bermain pada game labirin 2D ini terletak pada penelusuran rute perjalanan yang harus dilalui oleh seorang pemain dalam menemukan jalur mana yang tepat. Objek digerakkan dengan menggunakan tombol pada *keyboard* dengan menekan tombol panah atas untuk menggerakan maju ke depan, tombol panah bawah untuk menggerakan mundur ke belakang, tombol panah kiri untuk menggerakan belok ke kiri,

tombol panah kanan untuk menggerakan belok ke kanan. Hal ini dilakukan sampai object menemukan jalan keluar [9].

B. *Algoritma Fisher-Yates Shuffle*

Fisher-Yates Shuffle, yang dinamakan sesuai dengan pengembangnya yaitu Ronald Fisher dan Frank Yates, ini digunakan untuk pengacakan posisi atau input (*list*). Posisi permutasi dihasilkan oleh algoritma ini muncul dengan propabilitas sama [10].

Algoritma Fisher-Yates Shuffle merupakan metode pengacakan yang lebih baik atau dapat dikatakan sesuai untuk pengacakan angka, dengan waktu eksekusi yang cepat serta tidak memerlukan waktu yang lama untuk melakukan suatu pengacakan. Algoritma *Fisher-Yates* terdiri dari dua metode yakni, metode orisinal dan metode modern [11].

Metode orisinal yang digunakan untuk menghasilkan permutasi secara acak untuk angka 1 sampai N sebagai berikut [11].

1. Tuliskan angka dari 1 sampai N.
2. Pilih sebuah angka acak K diantara 1 sampai dengan jumlah angka yang belum dicoret.
3. Dihitung dari bawah, coret angka K yang belum di coret, dan tuliskan angka tersebut di lain tempat.
4. Ulangi langkah 2 dan langkah 3 sampai semua angka sudah tercoret.
5. Urutan angka yang dituliskan pada langkah 3 adalah permutasi acak dari angka awal.

Pada versi modern yang digunakan sekarang, angka terpilih tidak dicoret, tetapi posisinya ditukar dengan angka terakhir dari angka yang belum terpilih. Berikut metode modern yang digunakan untuk menghasilkan permutasi acak untuk angka 1 sampai N [11].

1. Masukkan array yang akan diacak urutannya.
2. Hitung jumlah elemen dari 1 sampai N.
3. Ambil elemen secara acak dari elemen yang tersisa.
4. Tukar dengan elemen ke-N.
5. Ulangi selama masih ada elemen yang tersisa.
6. Tampilkan array baru yang telah di acak urutannya.

Tabel 1 merupakan operasi algoritma Fisher-Yates Shuffle modern dengan ketentuan *range* adalah jumlah angka yang belum terpilih, *roll* adalah angka acak yang terpilih, *scratch* adalah daftar angka yang belum terpilih, dan *result* adalah hasil permutasi yang akan didapatkan [10].

Tabel 1. Contoh pengerjaan algoritma Fisher Yates Shuffle modern

Range	Roll	Scratch	Result
		12345678	
1-8	5	1234867	5
1-7	3	127486	3 5
1-6	4	12768	4 3 5
1-5	5	1276	8 4 3 5
1-4	2	167	2 8 4 3 5
1-3	3	16	7 2 8 4 3 5
1-2	1	6	1 7 2 8 4 3 5
Hasil Pengacakan :			6 1 7 2 8 4 3 5

C. *Algoritma Flood Fill*

Langkah yang paling tepat untuk dapat mengerti algoritma Flood Fill adalah dengan menggunakan analogi air yang ditumpahkan pada sebuah *maze*. Berikut langkah-langkah pengerjaan algoritma Flood Fill [6].

1. Proses penumpahan air terpusat hanya pada satu titik (*center*, selanjutnya titik ini akan dikenal sebagai tujuan).
2. Air akan membanjiri titik *center* ini, kemudian mulai mengalir ke area disekitarnya, yang tidak terhalang oleh dinding (dapat diakses secara langsung).
3. Secara virtual, *maze* dibagi menjadi beberapa kotak kecil (*array*).
4. Kotak di mana titik *center* berada, diberi nilai '0'.
5. Kotak yang terisi air setelah *center*, akan diberi nilai.
6. Kotak yang terisi air setelah golongan 1, akan diberi nilai 2.
7. Kotak yang terisi air setelah golongan 2, akan diberi nilai 3.
8. Dan begitu pula untuk kotak yang terisi air selanjutnya.

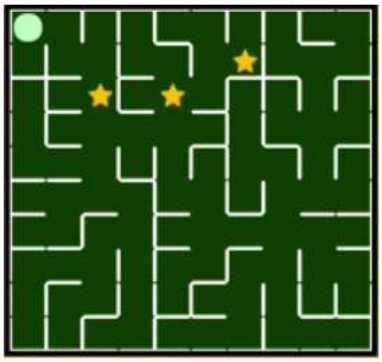
Arti dari nilai di dalam masing-masing kotak adalah jumlah kotak yang harus ditempuh dari kotak tersebut untuk mencapai *center* (tujuan). Asumsikan kotak yang berada pada bagian bawah sebelah kiri merupakan *start*, kemudian ikutilah kotak yang memiliki nilai lebih kecil dari nilai kotak yang sedang ditempati. Rute yang akan terbentuk adalah rute terpendek yang dapat ditempuh dari *start* menuju ke *center* [6].

D. *Perfect Maze*

Perfect maze adalah salah satu kategori labirin berdasarkan pola bentuk jalan yang ada di dalam labirin. Suatu labirin dikatakan *perfect maze* apabila kondisi-kondisi berikut terpenuhi [4]:

- Tidak ada jalan yang berulang atau memutar,
- Tidak ada sel yang terisolasi,
- Hanya memiliki 1 solusi.

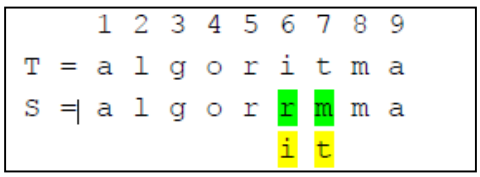
Gambar 1 merupakan contoh *perfect maze* yang seluruh kondisi terpenuhi.



Gambar 1. Contoh *perfect maze*

E. Hamming Distance

Algoritma Hamming Distance digunakan untuk mengukur jarak antara dua string yang ukurannya sama dengan membandingkan karakter kedua string pada posisi yang sama [12]. Gambar 2 menunjukkan contoh penggunaan Hamming Distance.

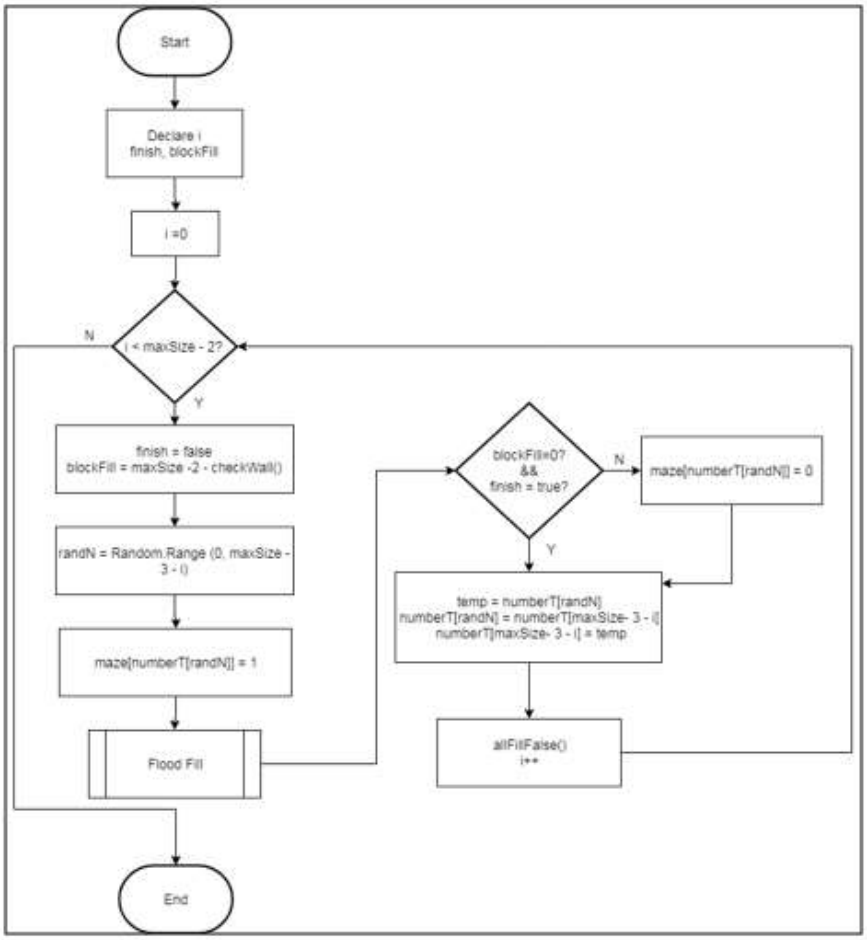


Gambar 2. Contoh penggunaan Hamming Distance

Dari Gambar 2, jarak yang didapat dengan menggunakan Hamming Distance adalah 2.

III. IMPLEMENTASI ALGORITMA

Pada penelitian ini, Algoritma Fisher-Yates Shuffle dan algoritma Flood Fill diimplementasikan pada *maze generator* pada permainan Labirin. Permainan Labirin yang dibangun adalah permainan *single player* yang cara memainkannya dengan menggunakan tombol panah untuk menggerakan karakter. Karakter berada dalam suatu labirin dan harus bergerak mencari jalan keluar.



Gambar 3. Flowchart penerapan algortima pada *maze generator*

Gambar 3 merupakan langkah-langkah penerapan algoritma pada *maze generator* permainan Labirin yang dijelaskan sebagai berikut.

1. Melakukan inisialisasi *block start*, *block finish*, dan seluruh *string* maze

Melakukan inisialisasi seluruh nilai *string* menjadi 0. Dilanjutkan pengacakan untuk membuat *block start* dan *block finish* pada *string maze*. Hasil pengacakan angka akan membuat nilai pada indeks *string* tersebut

menjadi 2 untuk Start dan 3 untuk Finish. Nilai pada indeks yang digunakan oleh *block start* dan *block finish* akan ditukar dengan nilai indeks terakhir agar nilai tersebut tidak muncul pada saat pengacakan algoritma Fisher-Yates Shuffle.

2. Pengacakan angka oleh algoritma Fisher-Yates Shuffle
   Melakukan pengambilan angka secara acak dengan *range* nol hingga total jumlah *block* yang nilainya belum muncul pada pengacakan sebelumnya.
3. Mengubah nilai pada *string maze* sesuai nilai pada indeks pengacakan
   String maze pada indeks hasil pengacakan angka akan diubah menjadi 1. Nilai 1 berarti *block* tersebut akan menjadi tembok.
4. Pengecekan oleh algoritma Flood Fill
   Hasil *string* setelah pengacakan akan diproses algoritma Flood Fill untuk pengecekan. Proses ini untuk melihat apakah seluruh *block* jalan dapat ditelusuri dan apakah *block finish* dapat dicapai.
5. Menetapkan nilai *string maze*
   Hasil algoritma *Flood Fill* untuk memastikan bahwa *string* pada indeks angka pengacakan tetap dapat menjadi tembok. Dari hasil algoritma *Flood Fill* jika seluruh jalan dapat ditempuh dan *block finish* dapat dicapai, maka nilai *string* indeks tersebut tetap 1. Sedangkan jika tidak terpenuhi, maka nilai *string* indeks tersebut kembali menjadi 0.
6. Menukar nilai indeks pengacakan dengan nilai indeks terakhir
   Setelah menetapkan nilai *string maze*, angka pada indeks pengacakan ditukar dengan angka indeks terakhir. Proses ini merupakan proses algoritma *Fisher-Yates Shuffle* setelah pengacakan angka. Proses ini dilakukan agar angka hasil pengacakan tidak muncul kembali pada proses pengacakan selanjutnya.
7. Melakukan langkah kedua sampai keenam hingga proses iterasi algoritma Fisher-Yates Shuffle selesai.
   Proses ini terus dilakukan hingga iterasi pada jumlah total *block maze*. Dapat dipastikan bahwa seluruh *block* dengan urutan acak dicoba untuk dijadikan tembok dan dilakukan pengecekan oleh algoritma *Flood Fill* agar *block* tersebut dinyatakan dapat atau tidak untuk menjadi tembok.
8. Mendapatkan hasil *string maze* dan membentuk *maze*
   Hasil proses algoritma *Fisher-Yates Shuffle* dan algoritma *Flood Fill* dalam bentuk *string maze*. Nilai pada *string* memiliki 4 arti, yaitu 0 sebagai jalan, 1 sebagai tembok, 2 sebagai *start*, dan 3 sebagai *finish*. Dari hasil *string* tersebut, akan dibentuk *maze* sesuai dengan ukuran level yang dipilih.

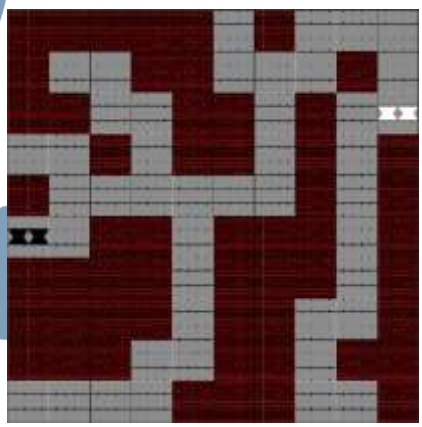
IV. IMPLEMENTASI DAN PEMBAHASAN

Gambar 4. Tampilan permainan labirin

Gambar 4 merupakan hasil implementasi algoritma Fisher-Yates Shuffle dan Flood Fill pada permainan Labirin.

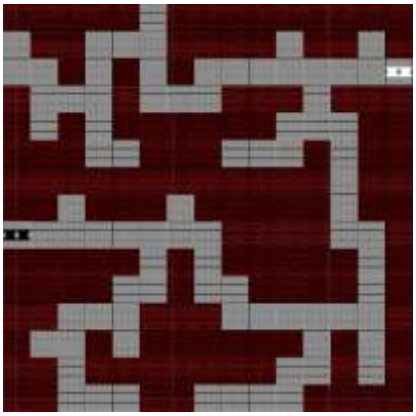
Empat menu utama dalam permainan Labirin adalah Start, How to Play, About, dan Exit. Terdapat tiga level yang dapat dipilih, yaitu Easy, Normal, dan Hard. Ukuran labirin akan dibentuk sesuai dengan level yang dipilih, yaitu:

- Tingkat Easy dengan ukuran 10 x 10 blok



Gambar 5. Contoh *maze* tingkat Easy

- Tingkat Normal dengan ukuran 15 x 15 blok



Gambar 6. Contoh *maze* tingkat Normal

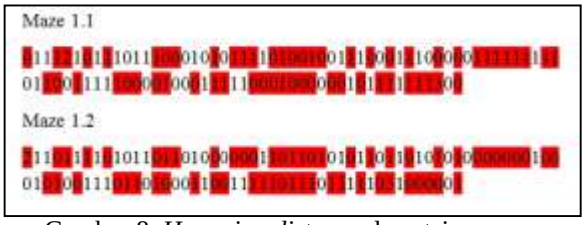
- Tingkat Hard dengan ukuran 20 x 20 blok

Gambar 7. Contoh *maze* tingkat Hard

Saat permainan telah dimulai, pemain akan menggerakkan karakter dengan menggunakan tombol anak panah untuk bergerak 4 arah. Pemain hanya bisa berjalan pada Floor (abu-abu) dan tidak bisa berjalan melewati Wall (merah bata). Permainan akan berakhir pada saat pemain menyentuh Finish.

V. EVALUASI

Uji coba aplikasi yang dilakukan adalah menghitung persentase perbedaan hasil *string* labirin yang telah dibuat dari hasil implementasi algoritma Fisher-Yates Shuffle dan algoritma Flood Fill menggunakan algoritma *Hamming Distance*. Hasil *string* labirin dibandingkan dengan *string* labirin yang di-*generate* pada *level* yang sama. Sebagai contoh Gambar 8 merupakan perbandingan dua *maze string* menggunakan *Hamming Distance*. Hasil perbedaan dari Maze 1.1 dengan Maze 1.2 menggunakan algoritma Hamming Distance adalah 60. Nilai ini digunakan untuk mengecek seberapa besar perbedaan Maze 1.1 dan Maze 1.2 yang dihasilkan oleh *maze generator*. Dari Gambar 8 dapat dilihat terdapat 60 perbedaan nilai yang ada pada Maze 1.1 dan Maze 1.2.



Gambar 8. *Hamming distance* dua *string maze*

Perhitungan algoritma *Hamming Distance* pada setiap *level maze* dilakukan secara manual. *Maze* yang diuji berjumlah 10 setiap *level* dan menghasilkan total 30 *maze* yang diuji. Hasil pengujian dapat dilihat pada Tabel 2.

Tabel 2. Hasil persentase perbedaan *maze*

Level	Rata-rata *Hamming Distance*	Jumlah *block*	Persentase perbedaan
Easy	48.78	100	48.78%
Normal	105.89	225	47.06%
Hard	192.6	400	48.15%
Rata-rata			48.00%

Berdasarkan Tabel 2, persentase perbedaan pada level Easy adalah 48.78%, persentase pada level Normal adalah 47.06% dan persentase perbedaan pada level Hard adalah 48.15%. Dari keseluruhan level, persentase yang didapat adalah 48%. Artinya perbedaan labirin setiap kali dimainkan adalah 48%.

Hasil labirin yang sudah didapat dilakukan pengecekan *perfect maze*. *Perfect maze* berarti labirin tidak memilki jalan yang memutar, tidak memilki sel yang terisolasi, dan hanya memilki 1 jalan keluar.

Tabel 3. Hasil pengecekan *perfect maze*

Maze	*Perfect Maze*
1.1	Ya
1.2	Ya
Maze	*Perfect Maze*
1.3	Tidak
1.4	Ya
1.5	Ya
1.6	Tidak
1.7	Ya
1.8	Ya
1.9	Ya
1.10	Ya
2.1	Ya
2.2	Ya
2.3	Ya
2.4	Ya
2.5	Ya
2.6	Ya
2.7	Ya
2.8	Ya
2.9	Tidak
2.10	Ya
3.1	Ya
3.2	Ya
3.3	Ya
3.4	Tidak
3.5	Ya
3.6	Ya
3.7	Ya
3.8	Ya
3.9	Ya
3.10	Tidak

Berdasarkan Tabel 3, bentuk labirin yang memilki bentuk *perfect maze* berjumlah 25 labirin dari 30 labirin yang diuji. Dengan demikian persentase labirin yang dibuat berbentuk *perfect maze* adalah 83.33%.

VI. SIMPULAN DAN SARAN

Hasil pengujian menggunakan algoritma *Hamming Distance* menunjukan bahwa setiap labirin yang dibentuk memiliki perbedaan. Rata-rata persentase perbedaan keseluruhan labirin yang dibentuk adalah

48%. Persentase *perfect maze* pada labirin yang dibentuk dengan algoritma ini adalah 83.33%.

Pengembangan penelitian selanjutnya dapat membuat *maze generator* dengan menggunakan algoritma lain dan menguji perbedaan *maze* yang dibentuk oleh *maze generator* algoritma tersebut. Penelitian dapat dilanjutkan dengan membandingkan suatu parameter dari *maze generator* penelitian ini dengan penelitian lainnya. Parameter yang dapat dibandingkan antara lain seperti kecepatan pembuatan *maze* atau persentase perbedaan setiap *maze*.

DAFTAR PUSTAKA

[1] Astawa, I Gede Santi, "Penggunaan Metode Kecerdasan Buatan Runut Maju dalam Memecahkan Permasalahan Game Labirin," Jurnal Ilmu Komputer, vol. 5, no. 1, hal. 37-46, 2012.

[2] Kurniawan, Meiki & Jumeilah, Fithri Selva, "Penerapan Algoritma Depth-First Search Sebagai Maze Generator pada Game Labirin Menggunakan Unity 3D," STMIK GI MDP, 2015.

[3] Street, Greg, dkk., "Random Map Generation in a Strategy Video Game," US Patent No. 7, 589, 742, Microsoft Corporation, 2009.

[4] Liga, T., Supriyono, S., & Selva Jumeilah, F., "Penerapan Algoritma Prim Sebagai Maze Generator Pada Game Labirin," STMIK GI MDP, 2016.

[5] Nugraha, Ryan, dkk., "Penerapan Algoritma Fisher-Yates Pada Aplikasi The Lost Insect Untuk Pengenalan Jenis Serangga Berbasis Unity 3D," STMIK GI MDP, 2015.

[6] Hendriawan, Akhmad., "Penyelesaian Jalur Terpendek dengan menggunakan Algoritma Flood Fill pada Line Maze," Industrial Electronic Seminar, 2010.

[7] Fikri, Hasnul Arief, dkk., "Perancangan Permainan Flood Filling pada Platform Android," Jurnal Dunia Teknologi Informasi, vol.1, no.1, hal. 35-43, 2012.

[8] Sutejoningtyas, F., "Perbandingan Algoritma Flood-Fill dengan Algoritma Backtracking dalam Pencarian Jalur Terpendek pada Robot Micromouse," Doctoral dissertation, Program Studi Teknik Elektro Fakultas Teknik Elektronika dan Komputer Universitas Kristen Satya Wacana), 2015.

[9] Sirait, Rina Br., "Perancangan Aplikasi Game Labirin dengan Menggunakan Algoritma Backtracking," STMIK Budidarma Medan, vol. 5, no. 2, hal. 100-103, 2013.

[10] Kurniawan, Fachrul, "Game Bahari Menggunakan Algoritma Fisher Yates Suffle Sebagai Pengacak Posisi NPC," Jurnal Ilmu Komputer dan Teknologi Informasi, vol. 7, no. 2, hal. 71-76, 2015.

[11] Zakifardan, I., "Implementasi algoritma dynamic weighting A* untuk pencarian rute terpendek pada NPC dan fisher-yates shuffle untuk pengaturan konten pada game 3D finding diamond," Doctoral dissertation, Universitas Islam Negeri Maulana Malik Ibrahim, 2016.

[12] Rochmawati, Y. dan Kusumaningrum, R., "Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks," Jurnal Buana Informatika, vol. 7, no. 2, hal. 125-134, 2016.