

**IMPLEMENTASI ALGORITMA DEPTH-FIRST SEARCH PADA
VIDEO GAME DENGAN GODOT ENGINE**

SKRIPSI

TRY JAKA GUNAWAN

171401048



PROGRAM STUDI ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

2023

IMPLEMENTASI ALGORITMA DEPTH-FIRST SEARCH PADA
VIDEO GAME DENGAN GODOT ENGINE

SKRIPSI

TRY JAKA GUNAWAN

171401048



PROGRAM STUDI ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

2023


PERSETUJUAN

Judul : IMPLEMENTASI ALGORITMA DEPTH-FIRST
SEARCH PADA VIDEO GAME DENGAN
GODOT ENGINE
Kategori : SKRIPSI
Nama : TRY JAKA GUNAWAN
Nomor Induk Mahasiswa : 171401048
Program Studi : SARJANA (S-1) ILMU KOMPUTER
Fakultas : ILMU KOMPUTER DAN TEKNOLOGI
INFORMASI UNIVERSITAS SUMATERA
UTARA


Medan, 2023

Komisi Pembimbing :

Pembimbing 2



Dr. Jos Timanta Tarigan, S.Kom., M.Sc
NIP. 198501262015041001

Pembimbing 1


Sri Mevani Hardi, S.Kom., M.Kom.
NIP. 198805012015042006

Diketahui/disetujui oleh

Program Studi S-1 Ilmu Komputer


Ketua
Dr. Analia, S.T, M.T
NIP. 197812212014042001

Universitas Sumatera Utara

PERNYATAAN**IMPLEMENTASI ALGORITMA DEPTH-FIRST SEARCH PADA VIDEO GAME
DENGAN GODOT ENGINE****SKRIPSI**

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing telah disebutkan sumbernya.

Sidikalang, 2023



Try Jaka Gunawan
171401048

PENGHARGAAN

1. Bapak Dr. Muryanto Amin, S.Sos., M.Si. selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia B.Sc., M.Sc., selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.
3. Ibu Dr. Amalia S.T., M.T. selaku Kepala Program Studi S-1 Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.
4. Ibu Sri Mevani Hardi, S.Kom., M.Kom. selaku Dosen Pembimbing I yang telah banyak memberikan bimbingan, saran dan dukungan dalam pengerjaan skripsi ini.
5. Bapak Jos Timanta Tarigan, S.Kom., M.Sc selaku Dosen Pembimbing II yang telah banyak memberikan bimbingan, saran dan dukungan dalam pengerjaan skripsi ini.
6. Bapak - selaku Dosen Penguji I yang telah banyak memberikan bimbingan, saran dan dukungan dalam pengerjaan skripsi ini.
7. Bapak - selaku Dosen Penguji II yang telah banyak memberikan bimbingan, saran dan dukungan dalam pengerjaan skripsi ini.
8. Seluruh tenaga pengajar dan pegawai Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara.
9. Orang tua tersayang Ayah Hasrian Rudi Rahimahullah, ibu Nurdiah, serta saudara Aprilla Ayu Andini, dan Dimas Lerian. Terima kasih untuk segala doa, nasihat dan kerja keras serta selalu memberikan dukungan semangat kepada penulis dalam menyelesaikan skripsi ini.
10. Teman-teman seluruh keluarga besar Kom C angkatan 2017 Ilmu Komputer Universitas Sumatera Utara yang telah banyak memberi motivasi kepada penulis dalam pengerjaan skripsi ini.
11. Teman-teman seperjuangan Mahasiswa Ilmu Komputer Universitas Sumatera Utara stambuk 2017 yang telah banyak memberi motivasi kepada penulis dalam pengerjaan skripsi ini.

12. Dan semua pihak yang telah banyak membantu dan mendukung penulis yang tidak bisa disebutkan satu per satu.

Sidikalang, 2023



Penulis

ABSTRAK

Permainan labirin adalah *genre* permainan yang menggunakan strategi untuk dimainkan. Jika *game developer* membentuk labirin secara manual, dibutuhkan upaya yang lebih dan bisa mengakibatkan ketertarikan pemain menurun. Pembuatan labirin bisa dibuat lebih baik dengan unsur *random*, dengan menggunakan algoritma *Depth-First Search* pembuatan labirin bisa dibuat dengan otomatis dan menghasilkan labirin acak yang unik setiap iterasi. Pada penelitian ini, permainan labirin ditambahi elemen lain untuk menambah ketertarikan pemain seperti *item* yang dapat menyegarkan atribut karakter pemain, dan perangkap maupun rintangan yang berinteraksi dengan karakter pemain sehingga menambah tingkat kesulitan permainan. Pada penelitian ini dikembangkan permainan labirin menggunakan algoritma *Depth-First Search* didalam *game engine* Godot dan didapatkan waktu pembuatan labirin bertumbuh dengan pertumbuhan *input* yang diberikan dan memiliki kompleksitas algoritma yang didapat $\theta(nm)$.

Kata Kunci : *Depth-First Search, Game Development, Labirin, Random, Godot.*

AN IMPLEMENTATION OF DEPTH-FIRST SEARCH ALGORITHM FOR A VIDEO GAME USING GODOT ENGINE

ABSTRACT

Maze game is a game genre which uses strategy to play. If game developer manually create the maze shape, more effort is needed and can cause player's interest to decreased. Maze creation can be improved with random element, with Depth-First Search algorithm maze creation can be automated and produce a unique random maze with every iteration. In this research, maze game we added another element to boost player's interest such as items that can refresh the player stats, and traps or obstacle which interacts with the player and give some level of difficulty to the game. In this research we develop maze game with Depth-First Search algorithm using Godot game engine and obtained that the time to generate the maze grow with the input given with an algorithm complexity of $\theta(nm)$.

Kata Kunci : Depth-First Search, Game Development, Maze, Random, Godot.

DAFTAR ISI

PERSETUJUAN	II
PERNYATAAN	III
PENGHARGAAN	IV
ABSTRAK	VI
ABSTRACT	VII
DAFTAR ISI	VIII
DAFTAR GAMBAR	X
DAFTAR TABEL	XI
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	3
1.3. Batasan Masalah	3
1.4. Tujuan Penelitian	4
1.5. Manfaat Penelitian	4
1.6. Metodologi Penelitian	4
1.7. Sistematika Penulisan	5
BAB 2 LANDASAN TEORI	7
2.1. Videogame	7
2.2. Permainan Labirin	7
2.3. Algoritma Depth-First Search	8
2.5. Kompleksitas Algoritma	13
2.5.1. Kompleksitas Waktu	14
2.6. Penelitian yang Relevan	15
BAB 3 ANALISIS DAN PERANCANGAN SISTEM	16
3.1. Analisis Sistem	16
3.1.1. Analisis Masalah	16
3.1.2. Analisis Kebutuhan	17
3.1.1. Analisis Proses	18

3.2. General Arsitektur Sistem	19
3.3. Pemodelan Sistem	21
3.4. Flowchart	25
3.4.1. Rancangan Halaman <i>Title</i>	25
3.4.2. Rancangan Halaman <i>Tutorial</i>	26
3.4.3. Rancangan Halaman <i>Menu</i> Interaktif	26
3.4.4. Rancangan Halaman <i>Level</i>	28
BAB 4 IMPLEMENTASI DAN PENGUJIAN	30
4.1. Implementasi Sistem	30
4.1.1. Tampilan Halaman Title	30
4.1.2. Tampilan Halaman Tutorial	31
4.1.3. Tampilan Halaman Menu	32
4.1.4. Tampilan Halaman Levels	33
4.2. Pengujian	34
4.2.1. Pengujian Implementasi Algoritma DFS pada labirin	34
4.2.2. Perhitungan Manual Algoritma Depth-First Search pada labirin	36
4.3. Hasil Pengujian	44
4.3.1. Pengujian mulai permainan	45
4.3.2. Pengujian evaluasi labirin	45
4.3.3. Kompleksitas Algoritma	46
BAB 5 KESIMPULAN DAN SARAN	49
5.1 Kesimpulan	49
5.2 Saran	49
DAFTAR PUSTAKA	50
LISTING PROGRAM	52

DAFTAR GAMBAR

Gambar 2. 1 Contoh Videogame labirin.....	18
Gambar 2. 2 Contoh Grafik Big- θ (theta).....	24
Gambar 3. 1 Diagram Ishikawa.....	27
Gambar 3. 2 General Arsitektur.....	30
Gambar 3. 3 Use Case Diagram.....	31
Gambar 3. 4 Activity Diagram.....	33
Gambar 3. 5 equence Diagram.....	34
Gambar 3. 6 Rancangan Halaman Title.....	35
Gambar 3. 7 Rancangan Halaman Tutorial.....	36
Gambar 3. 8 Rancangan Halaman Menu.....	37
Gambar 3. 9 Rancangan Dalam Level.....	39
Gambar 4. 1 Halaman Title.....	41
Gambar 4. 2 Halaman Tutorial.....	42
Gambar 4. 3 Halaman Menu.....	43
Gambar 4. 4 Halaman Level (Inti Permainan).....	44
Gambar 4. 5 Halaman Level (Zoom out).....	44
Gambar 4. 6 Hasil pengujian algoritma depth-first search menggenerasi labirin.....	46
Gambar 4. 7 Id ubin serta pasangan ubinnya.....	48
Gambar 4. 8 Hasil Pengujian Algoritma DFS pada Editor Godot.....	55

DAFTAR TABEL

Tabel 4. 1 Hasil Elapsed Time Algoritma Depth-First Search.....	48
Tabel 4. 2 Pengujian Mulai Permainan.....	56
Tabel 4. 3 Kompleksitas Algoritma Depth-First Search.....	57

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Video game adalah sarana permainan elektronik yang bertujuan untuk menghibur pemain dan juga melatih kemampuan kognitif pemain. Maka itu banyak pemain dari kalangan muda maupun tua menggunakan video game sebagai penghilang kebosanan, olahraga otak, maupun relaksasi beban mental. *Genre- genre video game* berkembang seiring dengan perkembangan teknologi dan inovasi(Lawson, 2019). Pada penelitian ini *genre* atau tema video game yang digunakan adalah tema labirin atau bisa juga disebut dengan *maze*.

Labirin atau *maze* adalah permainan dimana pemain berusaha untuk mencapai tujuan dengan cara mencari jalur yang mengantarkan ke jalan keluar ataupun suatu tujuan lainnya dari labirin yang terbentuk dari liku jalur yang dibatasi oleh dinding yang tidak bisa dilewati oleh pemain. *Gotcha(1973)* adalah *video game* pertama yang bertemakan labirin walaupun kepopulerannya dibayangi oleh *video game Pac-man(1980)*. Proses pembuatan *maze* adalah proses mendesain posisi jalur dan dinding dalam *maze* tersebut(Hendrawan, 2018).

Pada umumnya permainan labirin memiliki labirin yang dibentuk secara rumit dan dengan waktu yang sepadan, seperti *Tomb of The Mask(2016)*, *Marble Madness(1984)* yang mengakibatkan pembuatan jalur labirin kurang efisien dan menambah kompleksitas setiap pembuatan jalur labirin, serta mengurangi ketertarikan pemain dengan desain labirin yang monoton. Penelitian ini memiliki tujuan untuk membuat desain labirin yang

senantiasa berubah dan mengurangi waktu dan upaya yang dibutuhkan dalam pembuatan labirin dengan mengimplementasikan suatu algoritma.

Maze atau labirin yang digunakan dalam penelitian ini adalah *maze* yang berbentuk persegi empat dengan Panjang dalam jarak 5-20 sel dan Lebar dalam jarak 5-20 sel. *Maze* yang dibentuk memiliki jalan buntu yang akan menjadi hambatan dan tantangan kepada pemain. Berbeda dengan *labyrinth* yang tidak memiliki jalan buntu, *labyrinth* memiliki tujuan untuk menunda pemain dari poin A ke poin B dengan mendisorientasi jalur lurus (Fernandez-Vara, 2007). *Maze* pada penelitian ini dibentuk secara *random* dengan algoritma Depth-First Search.

Labirin yang digunakan pada penelitian ini berbentuk multikursal yang berarti mempunyai jalur bercabang, dan mempunyai jalan buntu yang memiliki peran sebagai hambatan dan tantangan kepada pemain. Labirin pada penelitian ini dibentuk secara *random* dengan algoritma Depth-First Search.

Algoritma pembuatan permainan labirin ada berbagai variasi seperti algoritma Depth-First Search, algoritma Binary Tree, algoritma Kruskal, algoritma Prim, algoritma Sidewinder dll. Metode seperti Depth-First Search bisa digunakan untuk membuat konten *video game* secara generasi prosedural (Zhang, Zeng, Dual, 2018).

Untuk penelitian ini algoritma yang digunakan adalah algoritma Depth-First Search karena hasil akhir labirin yang sesuai dengan konsep *video game* yang dikembangkan untuk penelitian ini. *Maze* yang digenerasi oleh algoritma Depth-First Search untuk penelitian ini

Metode pembuatan *video game* ada berbagai macam, salah satunya adalah menggunakan *game engine*. Godot adalah *game engine* gratis dan

mempunyai komunitas yang bersifat *open-source*. Godot dilisensi oleh *MIT (Massachusetts Institute of Technology)* yang ditujukan untuk memudahkan pelajar untuk membangun *video game* tanpa mengkhawatirkan tentang biaya karena perangkatnya yang bersifat gratis. Pihak *MIT*, yaitu pemilik *engine* ini, tidak memungut biaya royalti jika pengembang *video game* memutuskan untuk memasarkan *video game* buatannya (Ricky, 2021).

Berdasarkan penjelasan yang baru diberikan penulis akan mengembangkan aplikasi *video game* labirin dua dimensi dengan Godot. *Video game* yang akan dikembangkan penulis akan menyediakan fitur-fitur tambahan yang akan membuat *video game* tidak monoton seperti, skor, objek pembantu pemain, perangkat yang menghalangi pemain.

Oleh sebab itu, dari latar belakang yang telah disampaikan maka penulis ingin melakukan penelitian dengan mengambil topik pembahasan *game development* yang mempunyai judul “Implementasi Algoritma Depth-First Search pada Videogame dengan Godot Engine” dengan algoritma yang telah dijelaskan diatas.

1.2. Rumusan Masalah

Permasalahan yang ingin diselesaikan dengan penelitian ini adalah bagaimana mengembangkan *video game* labirin berbentuk multikursal secara otomatis dan senantiasa berubah dengan algoritma Depth-First Search yang memiliki *item* penambah stat karakter pemain, perangkat yang mengurangi nyawa karakter pemain, rintangan berupa *timer* yang membatasi skor pemain, dll. Pada *video game* maze arcade.

1.3. Batasan Masalah

Batasan masalah yang ditetapkan pada penelitian ini adalah :

1. Permainan dimainkan dengan kontrol pergerakan karakter pemain keatas, kebawah, kekiri, dan kekanan.

2. Pembuatan labirin menggunakan algoritma Depth-First Search.
3. Pembuatan perangkat pemain secara random
4. Pembuatan *item* pembantu pemain.
5. Perhitungan *running time* algoritma Depth-First Search dalam menggenerasi *maze* menggunakan kompleksitas algoritma
6. Permainan menggunakan grafik 2 dimensi.
7. *Game engine* yang digunakan adalah *Godot* dengan menggunakan bahasa pemrograman GdScript yang berbasis *Windows*.

1.4. Tujuan Penelitian

Tujuan penelitian yang ditetapkan pada penelitian ini adalah :

1. Mengimplementasikan algoritma Depth-First Search untuk membuat labirin yang akan dilalui oleh karakter pemain.
2. Menghitung kompleksitas algoritma Depth-First Search dengan menggunakan tabel kompleksitas.
3. Mengembangkan *video game* yang komprehensif dan menyenangkan dengan menambahkan fitur-fitur yang sudah di ringkas oleh penulis.

1.5. Manfaat Penelitian

Penelitian ini dapat memberikan manfaat berupa:

1. Penelitian diharapkan dapat menambah pengetahuan penulis dalam pengimplementasian algoritma Depth-First Search untuk membangun labirin dengan menggunakan *game engine Godot*.
2. Penelitian diharapkan dapat bermanfaat untuk proses *Game Development* dan sebagai referensi pada topik terkait.

1.6. Metodologi Penelitian

Penelitian ini menerapkan beberapa metode penelitian sebagai berikut :

1. Studi Pustaka

Pada tahap penelitian ini dimulai dengan mencari referensi terkait dari bermacam-macam sumber terpercaya serta melakukan kajian pustaka melalui artikel-artikel ilmiah, serta penelitian yang mempunyai relevansi dengan Algoritma Depth-First Search, *Recursive Backtrack*, *Random Number Generator*, Godot Engine.

2. Analisis dan Perancangan

Penulis membuat analisa untuk mencari tahu apa saja kebutuhan pada penelitian dan merancang diagram yang dibutuhkan seperti flowchart, use case diagram, dll.

3. Implementasi

Dalam tahap ini dilakukan implementasi sistem yang sudah dirancang ke dalam Godot Engine dengan bahasa pemrograman GDScript.

4. Pengujian

Pengujian dilakukan dalam tahapan ini terhadap *video game* yang telah dirancang.

5. Dokumentasi

Pada tahapan ini penulis melakukan dokumentasi dari tahap analisa sampai dengan tahap pengujian pada penelitian dalam bentuk skripsi.

1.7. Sistematika Penulisan

Penelitian ini menerapkan sistematika penulisan yang dijelaskan sebagai berikut :

BAB I : PENDAHULUAN

Dalam bab ini terdapat penjelasan latar belakang permasalahan yang dibahas pada penelitian, dan juga penjelasan rumusan masalah, batasan

masalah, tujuan penelitian, manfaat penelitian, metode penelitian, dan sistematika penulisan skripsi.

BAB II : LANDASAN TEORI

Pada bab ini terdapat tinjauan teoritis dari algoritma yang digunakan, dan penjelasan tambahan yang diperlukan pada penelitian ini.

BAB III : ANALISIS DAN PERANCANGAN

Bab ini menjelaskan tentang arsitektur umum penelitian sistem serta analisis masalah yang digunakan untuk menganalisis kebutuhan pembangunan sistem penelitian.

BAB IV : IMPLEMENTASI DAN PENGUJIAN

Dalam bab ini terdapat proses implementasi hal-hal yang digunakan pada pembangunan sistem serta pengujian masalah yang dinyatakan sebelumnya.

BAB V : KESIMPULAN DAN SARAN

Dalam bab ini terdapat kesimpulan yang diuraikan penulis dari pembahasan bab-bab sebelumnya dan saran sebagai masukan untuk permasalahan yang timbul dan diharap dapat membantu dalam penyelesaian masalah tersebut

BAB 2

LANDASAN TEORI

2.1. Videogame

Videogame adalah simulasi permainan yang di program menggunakan komputer dan bisa di interaksi oleh *user* dengan perangkat *input* (*joystick, keyboard, touchscreen, dll*) dan diterima oleh *user* melalui perangkat *output* (*monitor, speaker, proprietary device, dll*). *video game* dibarengi dengan dipasangnya komputer dengan *video display* (monitor), yang sepertinya menjadi observasi yang sudah jelas namun kirtikal dalam pembahasan tentang medium *video game* (Brodie, 2002).

2.2. Permainan Labirin

Permainan labirin adalah permainan yang mengharuskan pemain untuk menelusuri labirin dari poin A ke poin B dengan tambahan objektif-objektif yang harus dilakukan dan atau rintangan yang harus dilalui. Pembuatan labirin dapat dilakukan dengan manual yaitu dengan meletakkan satu persatu dinding labirin tetapi labirin bisa juga di generasi dengan algoritma, pada penelitian ini pembuatan labirin menggunakan algoritma Depth-First Search.



Gambar 2. 1 Contoh Videogame labirin
(https://play.google.com/store/apps/details?id=com.maze_squirrel.paid&hl=ln&gl=US)

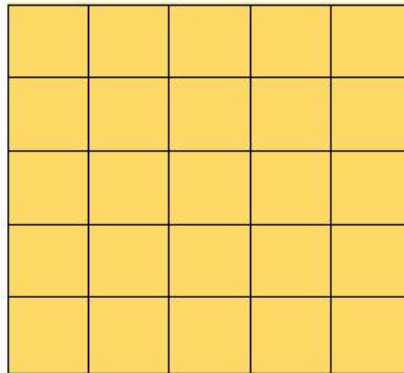
2.3. Algoritma Depth-First Search

Dalam penggenerasian labirin algoritma Depth-First Search adalah algoritma penjelajah pohon yang menjelajahi setiap anak pohon yang terjangkau dari induk pohon yang bersangkutan (Holzman, Peled, Yannakakis, 1996). Algoritma Depth-First Search adalah algoritma pencarian tanpa informasi yang menggunakan metode *brute force* (Lumenta, 2014). DFS lebih cocok untuk skenario di mana solusi terletak jauh dari sumbernya, menjadikannya pilihan efektif untuk masalah permainan atau teka-teki. Algoritme membuat keputusan dan mengeksplorasi semua jalur yang berasal dari keputusan tersebut (Iyanda J. 2018). Cara kerja algoritma Depth-First Search dalam pembuatan *maze* adalah dengan menentukan satu node awal yang dari node tersebut dimulai memahat jalur secara acak dari node tetangga atau sebelahny, tahap ini diulangi pada setiap node yang masih memiliki tetangga yang belum pernah dilalui. Jika node yang dipahat tidak lagi

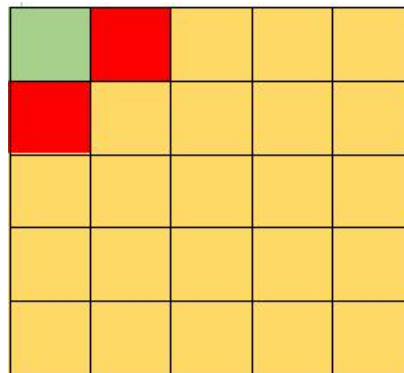
memiliki tetangga maka algoritma Depth-First Search akan menelusuri mundur akan node-node yang telah diapahat sampai menemukan node yang masih memiliki tetangga yang belum dikunjungi. Kemudian tahap-tahap tersebut diulangi sampai tidak ada lagi node yang belum dikunjungi.

Tahapan cara kerja algoritma Depth-First Search adalah dengan menyimpan node yang belum dikunjungi dan membandingkan dengan tetangga node yang sedang dikunjungi. Berikut adalah tahapan algoritma Depth-First Search dalam generasi labirin 5x5:

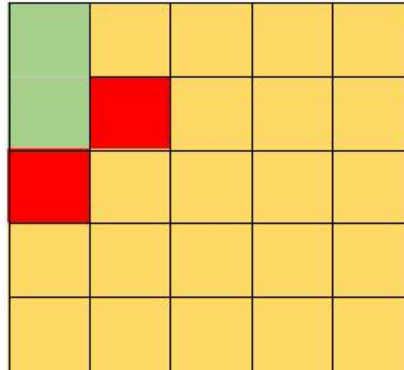
1. Inisialisasi tabel sel 5x5 tanpa jalur.



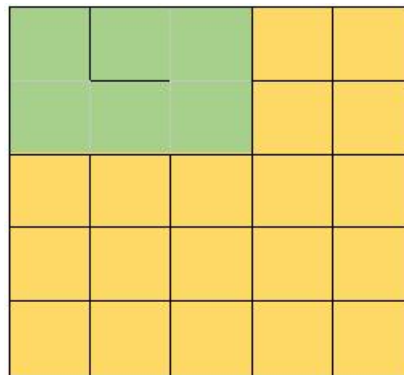
2. Dimulai dari sel sekarang (0,0) diperiksa ada berapa sel yang bertetangga dengan sel sekarang dan apakah sel sudah pernah dikunjungi atau tidak.



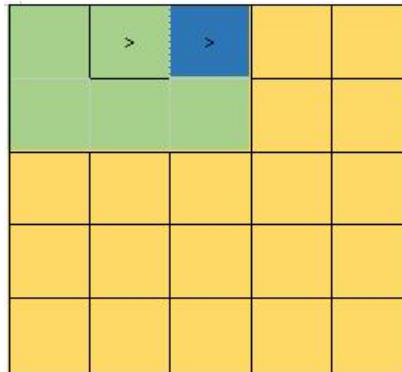
3. Dipilih satu sel secara acak dari semua sel tetangga. Jika sel yang dipilih adalah sel yang belum pernah dikunjungi maka pahat jalur kearah sel tersebut.



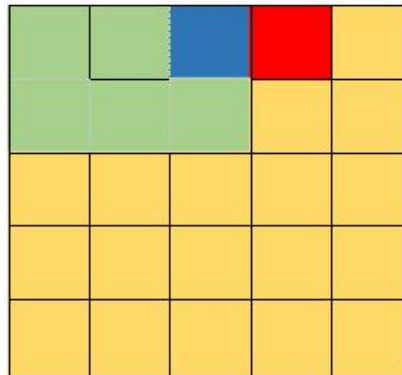
4. Ulangi tahap ke-3 sampai kondisi sel bertetangga dan belum dikunjungi tidak lagi terpenuhi.



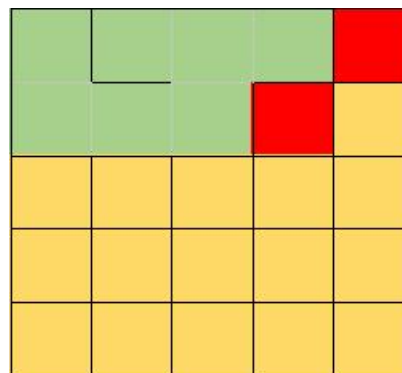
5. Jika sel sekarang tidak mempunyai tetangga yang belum dikunjungi. Maka diperiksa secara mundur apakah sel sebelumnya mempunyai tetangga yang belum dikunjungi.



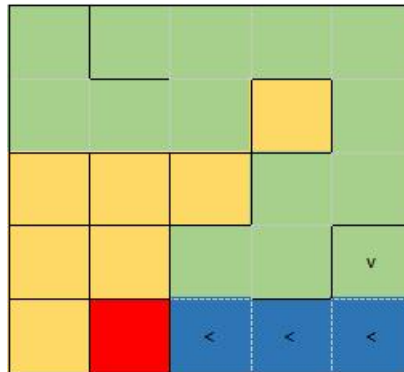
6. Ulangi tahap ke-5 sampai mendapatkan sel yang mempunyai tetangga yang belum dikunjungi.



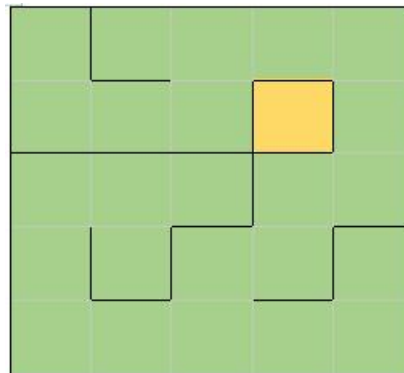
7. Bentuk labirin saat ini.



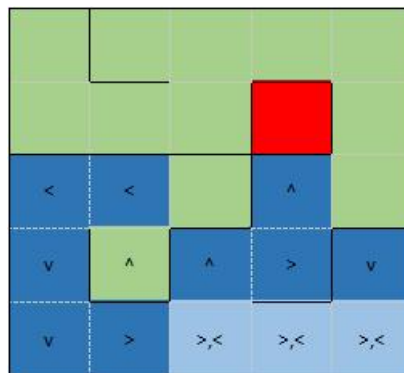
8. Bentuk labirin saat ini perulangan tahap ke-5.



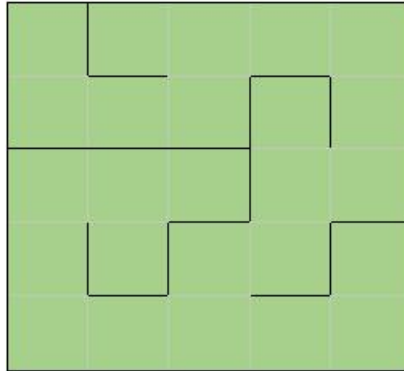
9. Bentuk labirin saat ini perulangan tahap ke-3.



10. Bentuk labirin saat ini perulangan tahap ke-5.



11. Algoritma selesai jika semua sel telah dikunjungi.



2.4. Random Number Generation (RNG)

Random Number Generation (RNG) adalah algoritma untuk mendapatkan angka secara acak. Secara realistis *RNG* adalah hasil dari simulasi algoritma deterministik yang jika diberikan suatu *input* akan memproduksi *output* yang bisa diprediksi dengan mempertimbangkan *input* yang dimasukkan. Tujuan dari algoritma tersebut adalah memproduksi rangkaian angka atau objek dengan mempunyai sifat yang nyaris tidak bisa dibedakan dengan rangkaian “*truly random*” untuk pengimplementasiannya pada aplikasi yang dibutuhkan (L’Ecuyer, 2012).

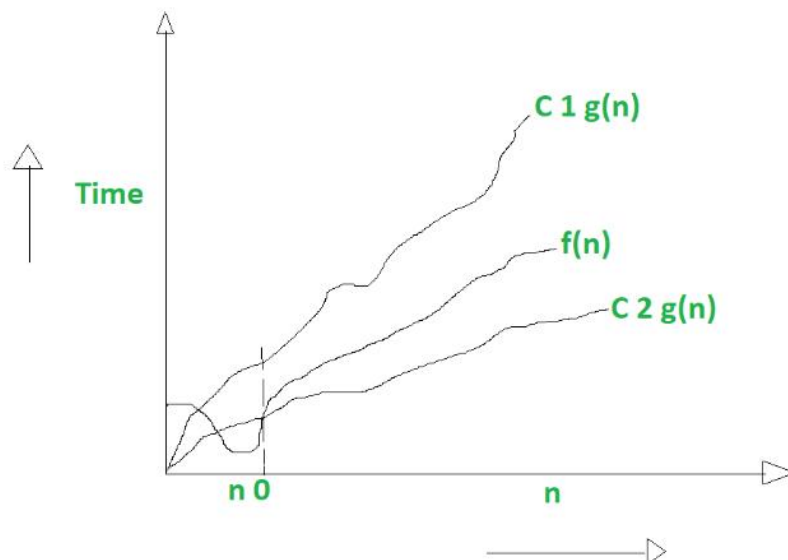
2.5. Kompleksitas Algoritma

Algoritma adalah metode untuk memecahkan berbagai permasalahan pada komputer dan kompleksitas algoritma adalah biaya yang dikenakan untuk memecahkan suatu masalah dengan suatu algoritma, diukur dengan running time, atau ukuran penyimpanan, atau unit relevan lainnya (Wilf, 2002).

2.5.1. Kompleksitas Waktu

Kompleksitas waktu sebuah algoritma adalah ukuran jumlah kalkulasi (komputasi) yang dioperasikan dengan komputer ketika memecahkan sebuah permasalahan dengan suatu algoritma. Ukuran yang dimaksud adalah jumlah waktu tempuh pemrosesan dan jumlah langkah komputasi (Nugraha, 2012). Pada penelitian ini digunakan notasi *Big-Θ* (theta) untuk mengkalkulasi kompleksitas waktu algoritma, yang bisa didefinisi dengan fungsi $t(n)$ adalah elemen dari $\Theta(g(n))$, apabila $t(n)$ dibatasi dengan konstanta positif c_1 dan c_2 dari atas dan bawah masing-masing maka

$$c_1 g(n) \leq t(n) \leq c_2 g(n) \quad \text{untuk } n \geq n_0$$



Gambar 2. 2 Contoh Grafik Big-θ (theta)

2.6. Penelitian yang Relevan

Penelitian yang disusun oleh penulis memiliki sejumlah penelitian terdahulu yang berkaitan yang disebutkan dibawah ini :

1. Penelitian yang sudah dilakukan oleh Lawson 2019 yang berjudul “Implementasi Metode Heuristic Dalam Game ELLER’S MAZE”. Penelitian ini mengimplementasikan algoritma backtracking pada permainan labirin.
2. Penelitian yang telah dilakukan oleh Zhang, Zeng, Dual 2018 yang berjudul “Procedural Content Generation for Room Maze”. Penelitian ini menyimpulkan bahwa algoritma Depth-First Search bisa digunakan untuk membuat konten video game secara prosedural
3. Penelitian yang telah dilakukan oleh Lumenta, 2014 yang berjudul “Perbandingan Metode Pencarian Depth-First Search, Breadth-First Search Dan Best-First Search Pada Permainan 8-Puzzle”. Penelitian ini menjelaskan bahwa algoritma adalah metode yang mengandalkan brute force.
4. Penelitian yang telah dilakukan oleh L’Ecuyer 2012 yang berjudul “Random Number Generator”. Penelitian ini menyimpulkan bahwa rng tidak sebenarnya acak, dan rng adalah hasil simulasi algoritma deterministik.
5. Penelitian yang telah dilakukan oleh Hendrawan 2018 yang berjudul “A Maze Game on Android Using Growing Tree Method”. Penelitian ini menjelaskan bahwa proses pembuatan maze adalah proses mendesain posisi jalur dan dinding.

BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1. Analisis Sistem

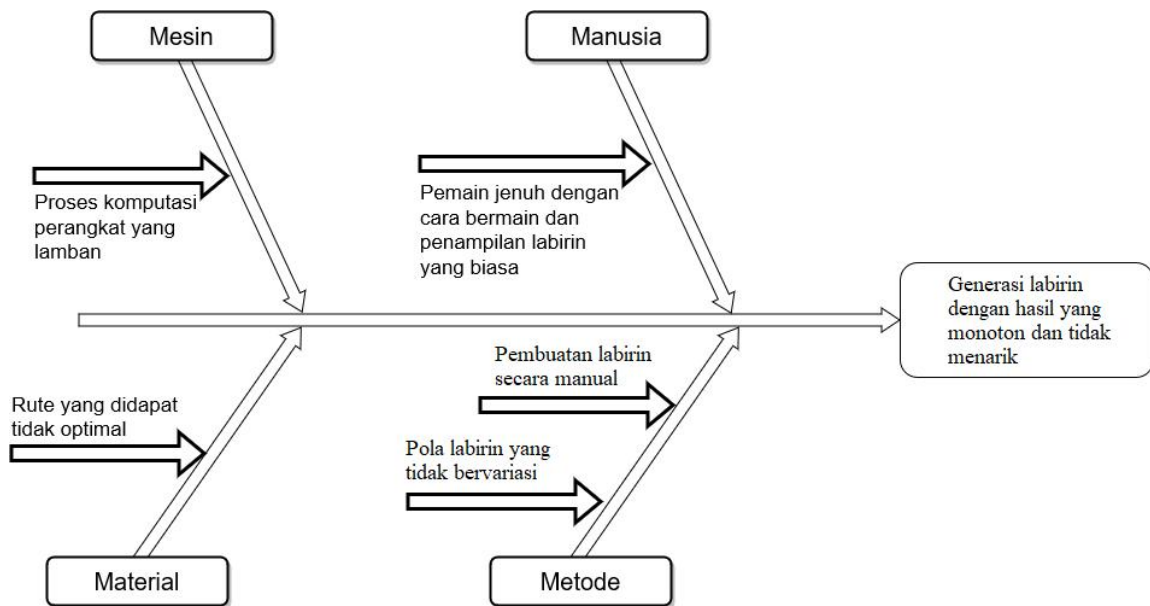
Analisis sistem adalah prosedur dalam perancangan produk yang dilakukan dengan membagi permasalahan menjadi komponen-komponen yang lebih mudah dipahami hubungannya antar bagian. Beberapa tahapan analisis sistem adalah analisis masalah dan analisis kebutuhan.

3.1.1. Analisis Masalah

Permainan labirin adalah permainan yang membutuhkan usaha dan waktu yang banyak untuk diwujudkan. Games atau permainan sendiri merupakan kegiatan hiburan yang diminati oleh setiap kalangan umur dari muda hingga tua tanpa terkecuali karena permainan ini sendiri merupakan kegiatan yang menghibur dan mengisi waktu luang serta bisa mengembangkan kemampuan koognitif pemain. Permainan yang masih diminati hingga kini adalah *video game*, salah satu jenis *video game* adalah puzzle. Namun, permainan puzzle ini masih dapat terus dikembangkan dengan mengadakan inovasi-inovasi yang unik pada permainan dengan genre lain seperti Adventure, Role-Playing dan lain-lain. Pada game ini pemain akan berusaha mencari jalan keluar dari labirin dengan bantuan hint.

Masalah utama dalam penelitian ini adalah cara mengimplementasi algoritma Depth-First Search untuk digunakan sebagai generasi labirin.

Identifikasi permasalahan pada penelitian ini pertama dilakukan dengan diagram Ishikawa atau diagram fishbone yang menjelaskan masalah yang dialami pada sistem serta sebab permasalahan yang terkait.



Gambar 3. 1 Diagram Ishikawa

3.1.2. Analisis Kebutuhan

Pada tahapan analisis kebutuhan ini didapat kebutuhan yang esensial antara lain kebutuhan fungsional dan kebutuhan non-fungsional.

i. Kebutuhan fungsional

Beberapa kebutuhan fungsional yang harus dipenuhi oleh sistem untuk menghadirkan sistem yang lengkap dan melayani user sesuai dengan tujuan sistem, yaitu:

- Sistem harus mampu menggenerasi labirin dengan algoritma *Depth-First Search*.
- Sistem harus mampu menerima *input* dari pemain untuk kontrol karakter di dalam *video game*.
- Sistem harus mampu menggenerasi *item* pembantu pemain dan bekerja dengan baik.
- Sistem harus mampu menggenerasi perangkat yang menghalau pemain secara *random*.

ii. Kebutuhan non-fungsional

Kebutuhan non-fungsional adalah kebutuhan yang menjelaskan batasan sistem seperti fitur, karakteristik, performa, dan basis sistem. Beberapa kebutuhan non-fungsional yang menjadi syarat kebutuhan yang harus dipenuhi sistem adalah:

- Aplikasi dapat dimainkan pada OS *Windows*.
- Aplikasi tidak memerlukan internet untuk dimainkan.
- Tampilan aplikasi dibuat menarik dan mudah digunakan *user*.
- Ditambah alur penyelesaian permainan agar meningkatkan interaksi dan ketertarikan pemain.

3.1.1. Analisis Proses

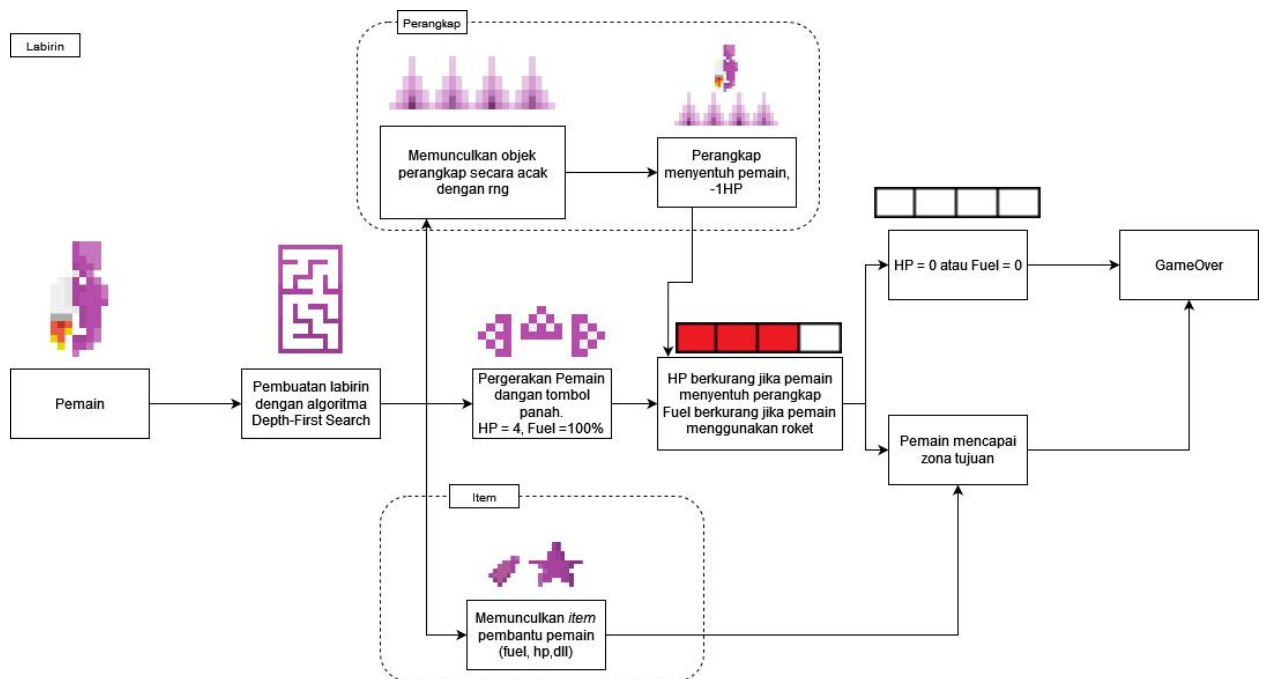
Algoritma yang diimplementasi pada sistem ini adalah algoritma *Depth-First Search* untuk membuat labirin.

Ketika pemain memulai permainan dengan masuk ke dalam sebuah *level* pada *menu* sistem, maka sistem akan menggenerasi labirin menggunakan algoritma *Depth-First Search*. Kemudian sistem menggenerasi karakter pemain serta memberikan kendali karakter kepada pemain, sistem juga menggenerasi *goal* dan *item* pembantu pemain, serta *trap* jika *level* yang dimainkan mempunyai *trap*.

3.2. General Arsitektur Sistem

Implementasi algoritma yang diajukan pada penelitian ini terdapat pada generasi labirin pada permainan. Algoritma Depth-First Search digunakan untuk membuat sebuah labirin yang akan dilalui oleh pemain.

Hal yang paling utama dalam *game* ini adalah pergerakan pemain. Pemain atau *user* adalah unsur paling penting yang membuat pemain menentukan berakhir atau tidaknya permainan. Pemain diberikan tujuan (*goal*) dan objek perangkap untuk menghalau perjalanan pemain dengan mengurangi *Health Point* (*HP*) pemain dan jika pemain sampai di *goal* atau *HP* pemain sama dengan 0 maka permainan akan berakhir. Pemain juga diberikan batasan dengan bentuk bahan bakar (*Fuel*) yang akan berkurang setiap kali pemain bergerak, dan ketika *Fuel* habis pemain tidak akan bisa bergerak sehingga dipaksa agar mengulangi permainan. *Timer* atau waktu batasan digunakan oleh sistem untuk merekam waktu yang ditampilkan sebagai skor pemain. Adapun arsitektur umum yang dirancang adalah sebagai berikut:



Gambar 3. 2 General Arsitektur

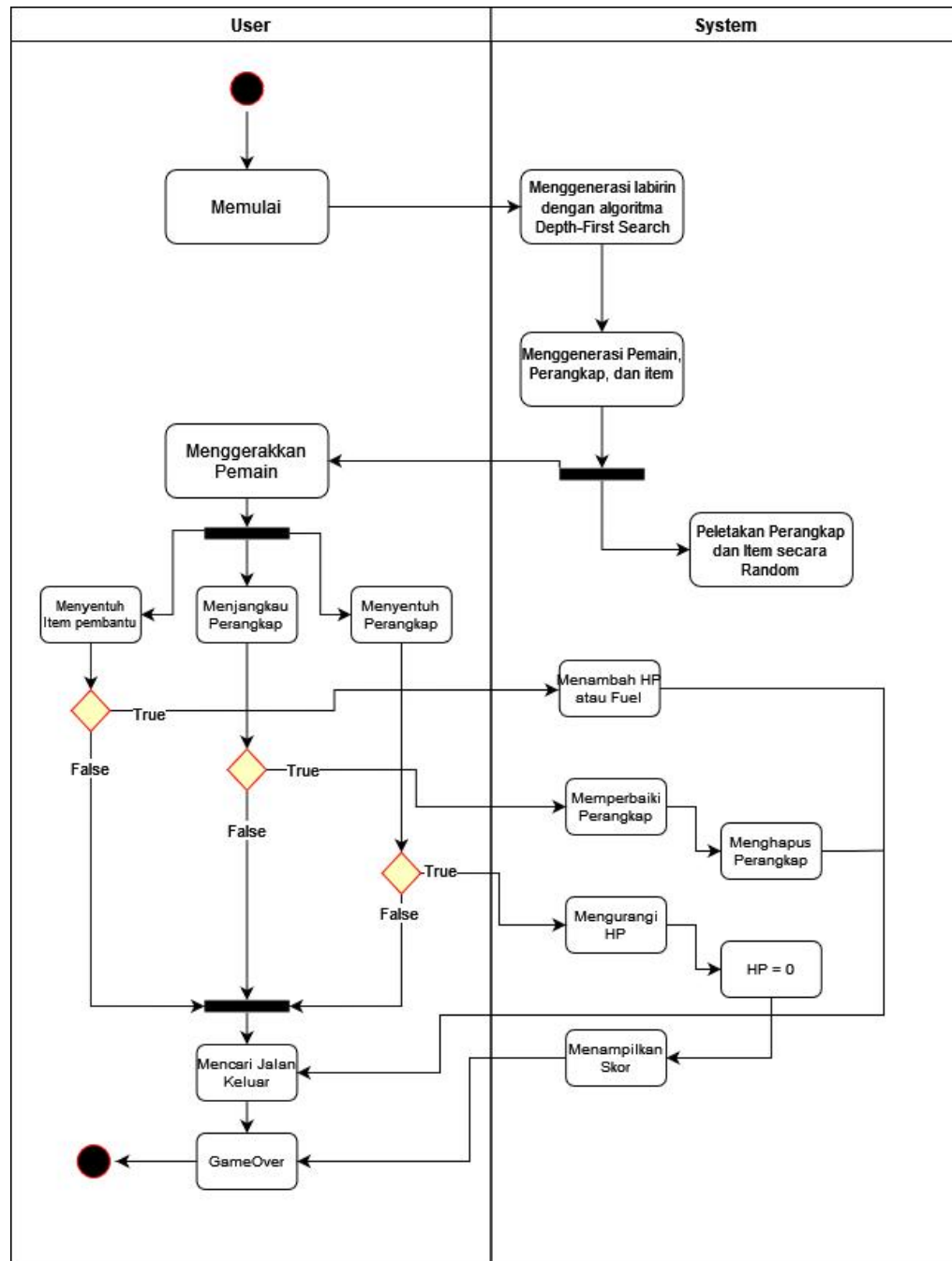
Tahapan pertama ketika pemain memulai permainan adalah pembuatan labirin dengan menggunakan algoritma Depth-First Search. Tujuan pembuatan labirin adalah agar pemain mempunyai permainan yang bisa dimainkan.

Kemudian tahapan selanjutnya adalah pergerakan pemain dengan tombol panah. Pemain adalah elemen penting dalam permainan ini, karena pemain yang menentukan permainan berakhir atau tidak. Pemain mencari jalan keluar dari labirin dengan disertai perangkap yang akan mengurangi *HP* pemain. Pemain juga bisa mengumpulkan *item* yang bisa membantu pemain dalam menyelesaikan permainan seperti *item_fuel_station* yang akan mengisi ulang *Fuel*.

Setelah itu pemain baru bisa menggerakkan karakter sesuai dengan *input* yang diberikan.

3.3.2. Activity Diagram

Activity Diagram dipakai sebagai penampilan aktivitas dan tindakan sistem. Gambaran dalam *activity diagram* menampilkan bagaimana sistem dimulai sampai sistem berakhir.



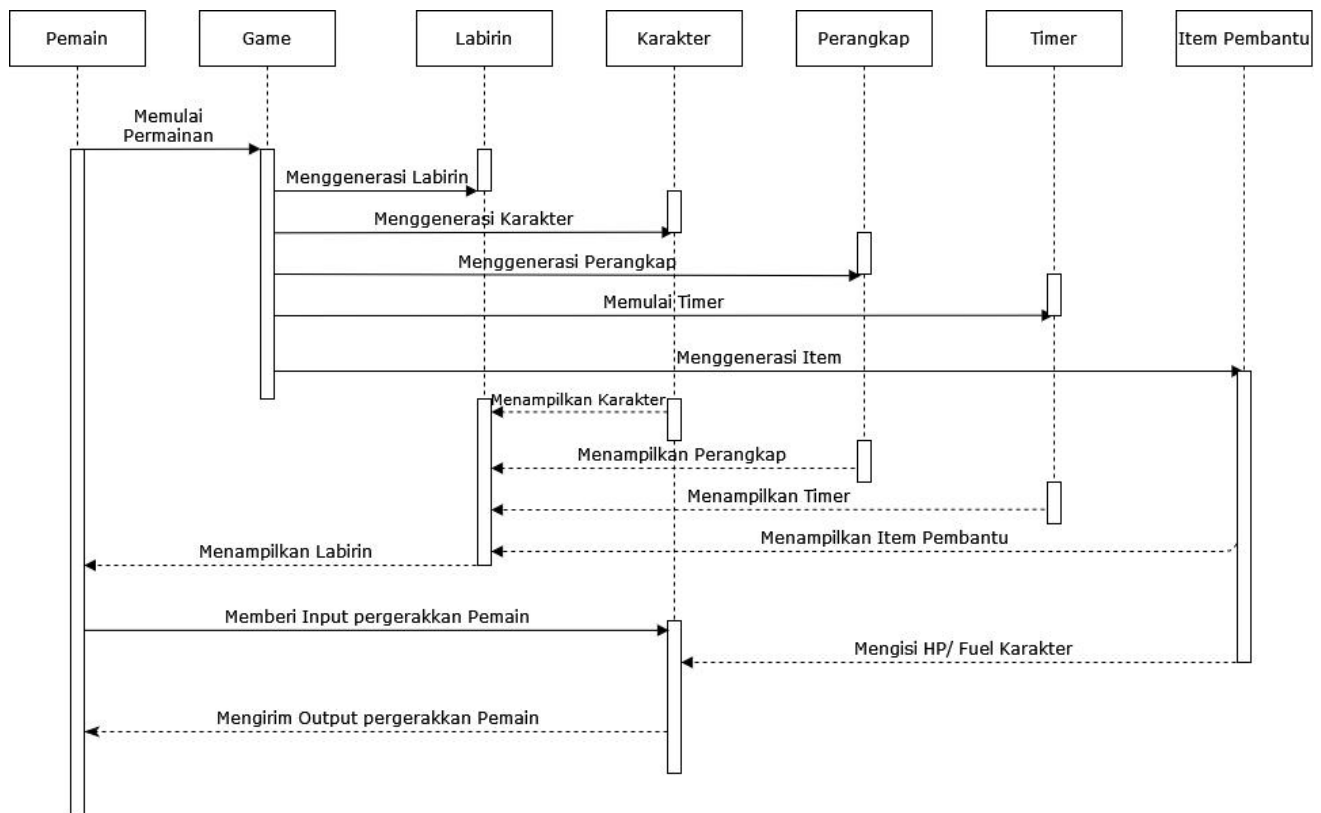
Gambar 3. 3 Activity Diagram

Pada gambar di atas dijelaskan interaksi aktivitas antara pemain dan sistem secara sistematis. Kolom kiri berisi aktivitas dari *user* yang berinteraksi

dengan aktivitas sistem yang berada di kolom kanan yang membalas dengan respons kepada aktivitas *user*.

3.3.3. Sequence Diagram

Sequence Diagram dipakai sebagai penampilan objek yang saling berinteraksi di dalam sistem yang disusun dalam sebuah rangkaian waktu.



Gambar 3. 4 equence Diagram

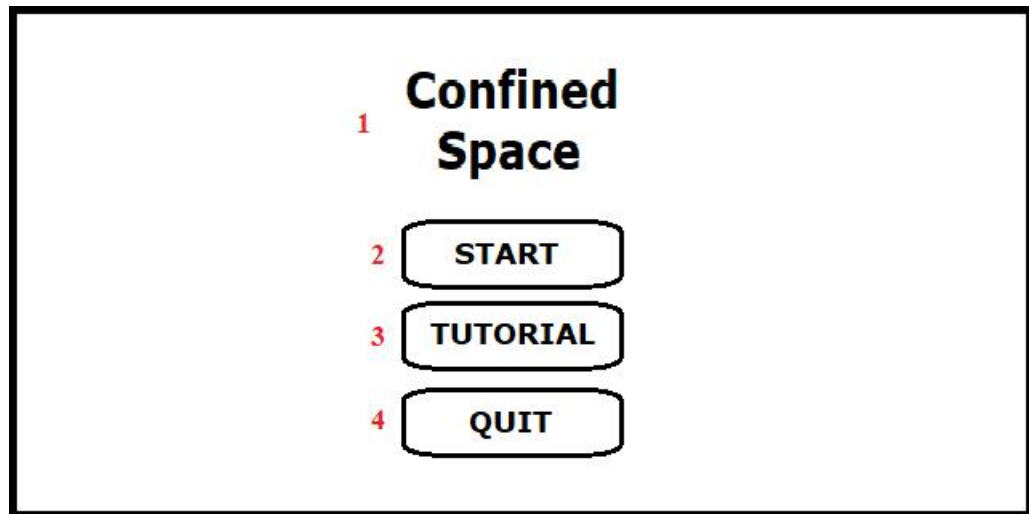
Gambar diatas menggambarkan interaksi *user* dengan pemain dalam bentuk diagram yang berurutan. Dimulai dari *user* yang menggambarkan interaksi dengan tanda panah garis penuh yang dibalas oleh sistem yang digambarkan oleh tanda panah garis putus-putus.

3.4. Flowchart

Flowchart adalah diagram yang merangkaikan alur pemecahan masalah yang digambarkan dengan grafik bermacam simbol yang terhubung dengan garis panah.

3.4.1. Rancangan Halaman *Title*

Halaman *Title* adalah halaman antarmuka yang ditemui pemain ketika membuka aplikasi permainan. Seperti gambar 3.8 , ada tiga pilihan yang diberikan sistem kepada pemain, yaitu: *Start*, *Tutorial*, dan *Quit*.



Gambar 3. 5 Rancangan Halaman Title

Keterangan gambar:

1. *Text* nama judul permainan.
2. *Button Start*, Tombol yang digunakan untuk mengakses *Menu*.
3. *Button Tutorial*, Tombol yang memunculkan halaman *Tutorial*.
4. *Button Quit*, Tombol yang digunakan untuk menutup sistem.

3.4.2. Rancangan Halaman *Tutorial*

Halaman *Tutorial* adalah halaman yang memberikan informasi kepada pemain bagaimana cara bermain dan penjelasan yang berkaitan yang bisa dilihat pada gambar 3.9 .



Gambar 3. 6 Rancangan Halaman Tutorial

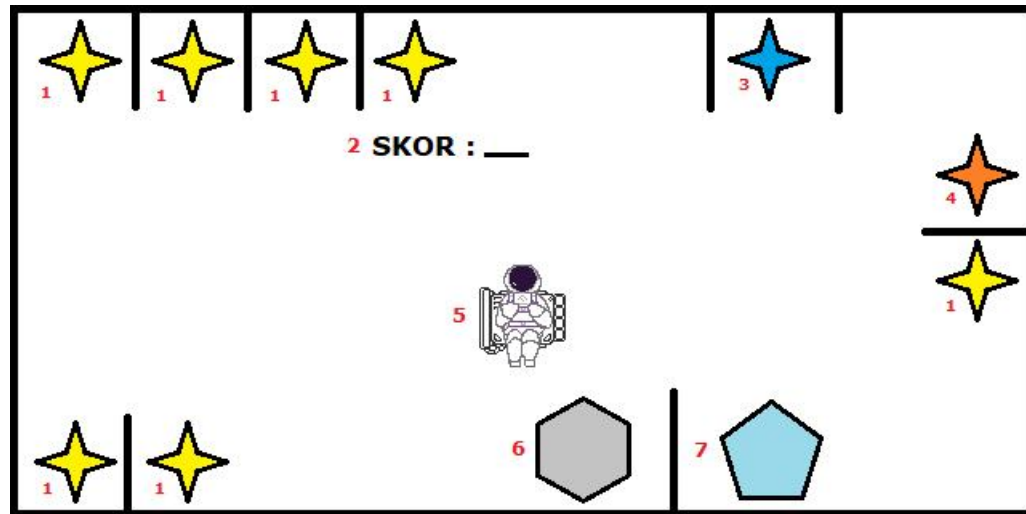
Keterangan gambar:

1. *Gambar*, mewakili objek-objek yang akan ditemui pemain di dalam permainan.
2. *Text*, berisikan penjelasan sesuai dengan gambar disampingnya.
3. *Button*, tombol yang digunakan untuk melihat halaman *Tutorial* selanjutnya.
4. *Button*, tombol yang digunakan untuk menutup halaman *Tutorial*.

3.4.3. Rancangan Halaman *Menu Interaktif*

Antarmuka *Menu* interaktif adalah halamant yang dimunculkan sistem ketika memilih *Start* pada halaman *Title*. Seperti yang dilihat pada gambar 3.10 pada halaman ini sistem menggenerasi objek seperti *LevelEntry*,

GameEnd, *TitleEntry*, *Skor*, *ItemUpgrade*, *ItemFuel*, dan sistem juga menggenerasi karakter pemain serta memberikan kendali kepada pemain.



Gambar 3. 7 Rancangan Halaman Menu

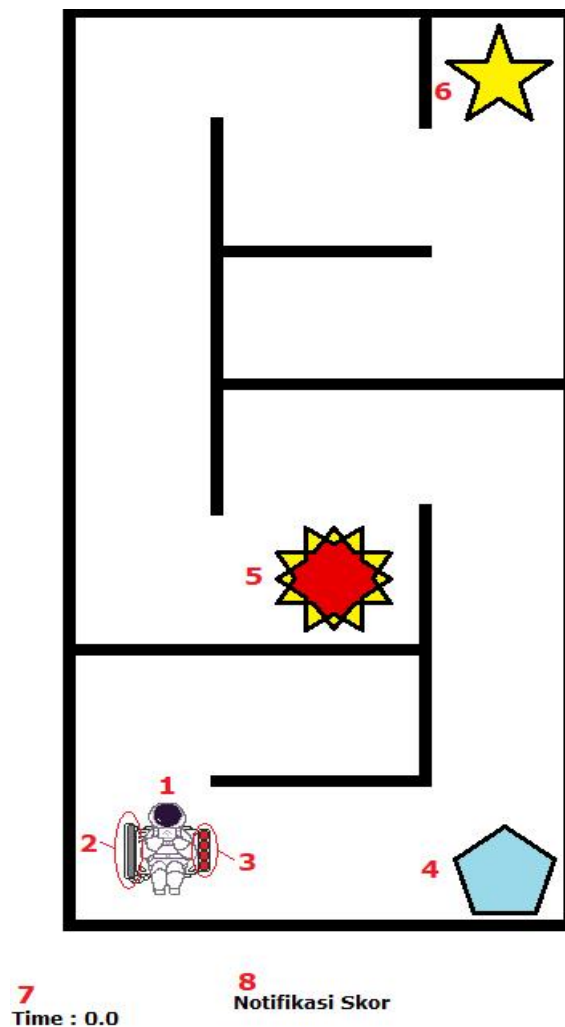
Keterangan gambar:

1. *LevelEntry*, merupakan objek yang digunakan pemain untuk mengakses *Level* yang sesuai.
2. *Text Skor*, *text* yang menampilkan skor yang sudah dikumpulkan pemain.
3. *TitleEntry*, merupakan objek yang digunakan pemain untuk kembali ke halaman *Title*.
4. *GameEnd*, merupakan objek yang digunakan pemain untuk mencapai tamat keseluruhan permainan.
5. *Karakter*, karakter yang dikendalikan pemain dalam permainan.

6. *ItemUpgrade*, merupakan objek yang dapat meningkatkan kemampuan pemain.
7. *ItemFuel*, merupakan objek yang dapat menambah *fuel* dan *HP* pemain.

3.4.4. Rancangan Halaman *Level*

Antarmuka Dalam *Level* adalah halaman antarmuka yang berisi inti permainan yang di generasi oleh sistem sesuai dengan rancangan setiap *level* yang bisa dilihat pada gambar .



Gambar 3. 8 Rancangan Dalam Level

Keterangan gambar:

1. *Karakter*, karakter yang dikendalikan pemain dalam *Level*.
2. *FuelIndicator*, adalah indikator seberapa banyak *fuel* pemain yang tersisa.
3. *HPIndicator*, adalah indikator seberapa banyak *HP* pemain yang tersisa.
4. *ItemFuel*, merupakan objek yang dapat menambah *fuel* dan *HP* pemain.
5. *Trap*, adalah objek rintangan pada *Level* yang bisa mengurangi *HP* pemain jika tersentuh pemain dan dapat dihilangkan oleh pemain jika pemain memperoleh peningkatan dari *ItemUpgrade* di *Menu*.
6. *Goal*, adalah objek tujuan yang dicari pemain untuk menyelesaikan *Level*.
7. *Text time*, *text* yang menampilkan waktu berjalan pada setiap *Level*.
8. *Text*, *text* yang menampilkan peringatan berapa Skor yang diperoleh ketika pemain menyelesaikan *Level*.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Sistem

Pada tahap implementasi, sistem pada penelitian ini dikembangkan pada sistem *hardware* i5-10500H dan menggunakan bahasa pemrograman GDScript, dalam *game engine* Godot. Sistem memiliki 4 tampilan halaman inti, yaitu: *Title*, *Tutorial*, *Menu*, dan *Levels*.

4.1.1. Tampilan Halaman Title



Gambar 4. 1 Halaman Title

Pada gambar 4.1 menampilkan halaman *Title*. Antarmuka berisikan judul permainan, tombol *Start*, tombol *Tutorial*, tombol *Quit*.

4.1.2. Tampilan Halaman Tutorial



Gambar 4. 2 Halaman Tutorial

Pada gambar 4.2 menampilkan halaman *Tutorial* yang berisikan arahan bermain dan penjelasan berbagai objek-objek yang akan dijumpai oleh pemain di permainan.

4.1.3. Tampilan Halaman Menu



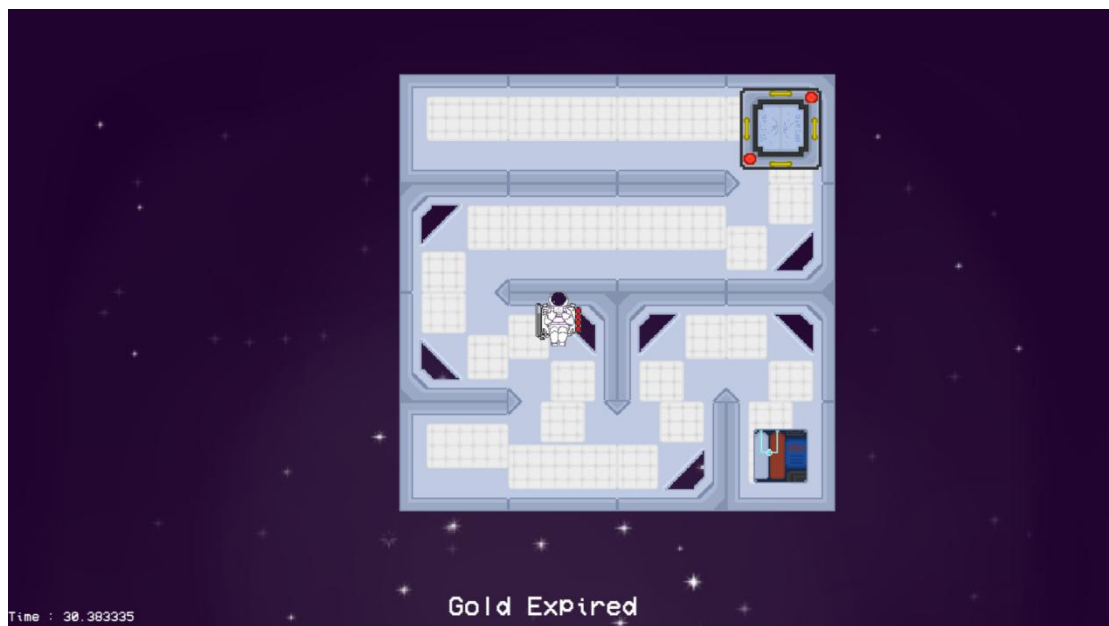
Gambar 4. 3 Halaman Menu

Pada gambar 4.3 menampilkan halaman *Menu* yang menunjukkan menu utama dimana pemain dapat mengendalikan karakter untuk menentukan *Level* mana yang akan dimainkan, apakah ingin kembali ke *Title*, meningkatkan kemampuan pemain, dan menamatkan keseluruhan permainan.

4.1.4. Tampilan Halaman Levels



Gambar 4. 4 Halaman Level (Inti Permainan)



Gambar 4. 5 Halaman Level (Zoom out)

Gambar 4.4 dan gambar 4.5 memperlihatkan bagaimana inti permainan dimainkan, karakter dikendalikan pemain dengan tanda panah di *keyboard* perangkat keras sistem. Terdapat teks yang mengandung informasi seperti *time*, notifikasi skor.

Untuk proses sistem yang berjalan pada inti permainan, ketika pemain memasuki suatu *Level*, sistem menggenerasi labirin dengan algoritma *Depth-First Search*, sistem juga menggenerasi karakter pemain, perangkap, item, dan *goal*. Ketika karakter pemain bergerak maju atau mundur *Fuel* karakter berkurang sedikit demi sedikit. Ketika karakter pemain menyentuh perangkap, *HP* karakter pemain akan berkurang sebanyak 1 poin. Jika kemampuan pemain sudah ditingkatkan dengan *ItemUpgrade* perangkap bisa dihilangkan dengan menekan tombol interaksi di *keyboard* perangkat keras sistem.

4.2. Pengujian

Pengujian sistem adalah proses menguji sistem yang telah dibangun untuk mencari tahu apakah sistem yang dibangun sudah memenuhi kebutuhan serta kriteria yang sesuai dengan perancangan sistem yang telah ditetapkan sebelumnya.

4.2.1. Pengujian Implementasi Algoritma DFS pada labirin

Pada penelitian ini, penggenerasi labirin menggunakan algoritma *Depth-First Search*. Permainan ini mempunyai 7 *Level* yang berbeda, hal yang membedakan setiap *Level* adalah ukuran, dan rintangan. Contohnya pada *Level* 1 ukuran labirin adalah 5 x 20, lalu pada *Level* 4 ukuran labirin adalah 20 x 5 dan mempunyai perangkap, dan lainnya.

Pada proses pengujian ini, labirin yang digunakan mempunyai ukuran 5 x 5. Gambar menampilkan perolehan uji algoritma *Depth-First Search* untuk menggenerasi labirin.



Gambar 4. 6 Hasil pengujian algoritma depth-first search menggenerasi labirin

Ketika pemain memulai *Level*, sistem menggenerasi labirin. Sel awal ditetapkan pada titik (0,0) atau sudut kiri bawah, dari sel tersebut dipilih secara acak sel tetangga dan jalur dipahat ke arah sel tetangga tersebut. Selanjutnya sistem menjalankan algoritma sampai semua sel sudah dikunjungi. Kemudian sistem meletakkan *tileset* atau ubin yang sesuai.

Pengujian dilakukan 10 kali percobaan yang ukurannya meningkat setiap percobaan untuk mencari tahu waktu yang diperlukan (*running time*) algoritma Depth-First Search dalam menggenerasi labirin yang bisa dilihat pada tabel 4.1.

Tabel 4. 1 Hasil Running Time Algoritma Depth-First Search

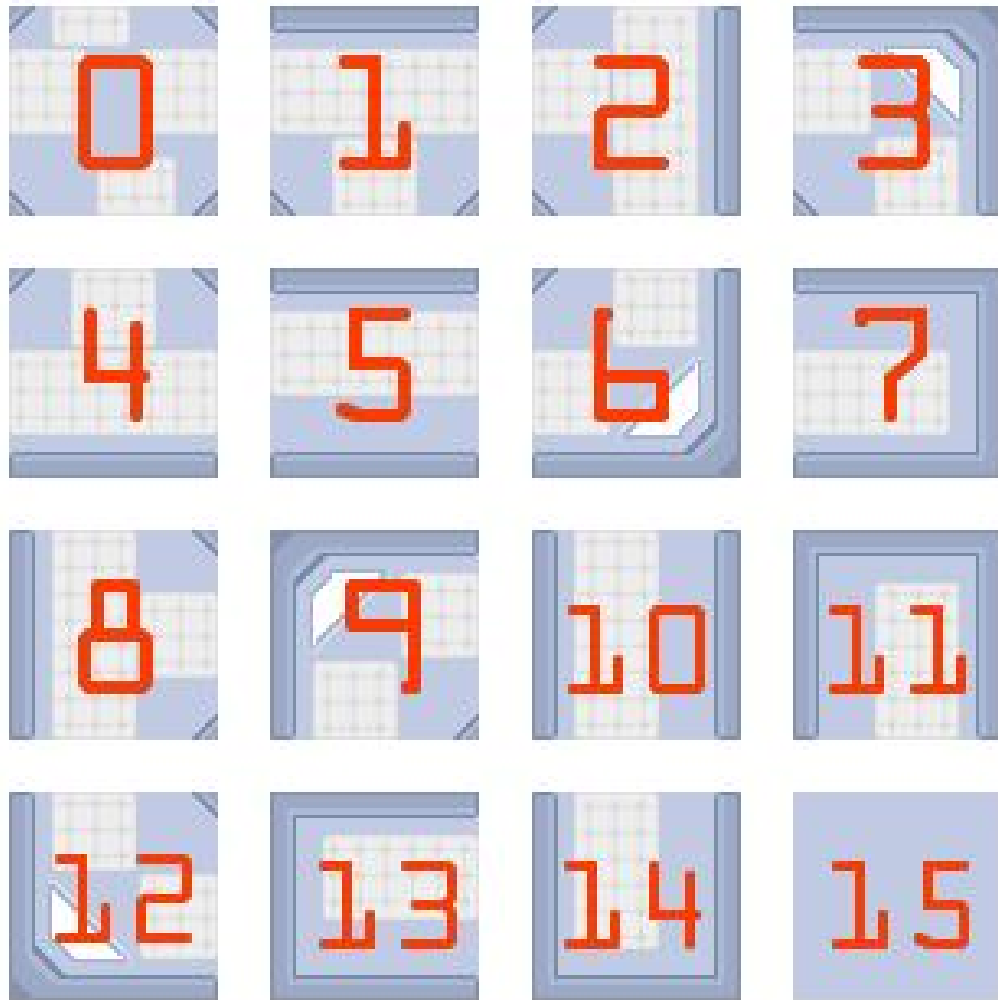
No	Ukuran	Estimasi Waktu (ms)
1	5 x 5	1,255
2	15 x 15	7,144
3	25 x 25	34,314
4	35 x 35	101,503
5	45 x 45	263,600
6	55 x 55	595,828
7	65 x 65	1116,194
8	75 x 75	2072,109
9	85 x 85	3085,156
10	95 x 95	5479,728

Dari tabel 4.1, dapat disimpulkan bahwa *running time* algoritma bertumbuh dengan ukuran *input* yang diberikan, dibuktikan dengan *running time* algoritma yang dibutuhkan bertambah lama pada setiap percobaan.

4.2.2. Perhitungan Manual Algoritma Depth-First Search pada labirin

Pada penelitian ini proses pembuatan labirin dengan algoritma *Depth-First Search* ditetapkan desimal dan koordinat *vector* untuk merepresentasikan arah labirin, dan label *id* untuk setiap ubin yang akan diletakkan. Arah [Utara, Timur, Selatan, Barat] masing-masing direpresentasikan dengan koordinat [(0,-1), (1,0), (0,1), (-1,0)] dan nilai [1, 2,

4, 8]. Berikut adalah perwakilan visual dari semua ubin yang memungkinkan.



Gambar 4. 7 Id ubin serta pasangan ubinnya

Perhitungan manual untuk labirin ukuran 5x5 dijelaskan dengan langkah-langkah sebagai berikut:

Iterasi 1

Inisialisasi sel 5x5 dengan meletakkan ubin *id*:15 pada setiap sel.

y/x	0	1	2	3	4
0	15	15	15	15	15
1	15	15	15	15	15
2	15	15	15	15	15
3	15	15	15	15	15
4	15	15	15	15	15

Sel awal dimulai dari (0,0) dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*. Kemudian arah ditentukan dengan memilih sel tetangga secara acak yaitu (1,0) dan menjumlahkan dengan rumus :

$$Dir = Next - Current$$

Dimana:

Dir = Arah

Next = Koordinat sel selanjutnya

Current= Koordinat sel sekarang

Menggunakan rumus tersebut maka didapat arah, $(1,0) - (0,0) = (1,0)$ yaitu Timur, kemudian agar mendapatkan *id* ubin yang tepat dikurangkan *id* ubin sel sekarang dengan nilai arah yaitu (Timur = 2) dengan hasil, 13, maka diletakkan ubin *id* 13 pada sel (0,0). Dihitung juga *id* ubin sel tetangga yang dipilih dengan mengurangkan *id* ubin selanjutnya dengan nilai (-)arah yaitu $((-1, 0) = \text{Barat} = 8)$ dengan hasil 7, maka diletakkan ubin *id* 7 pada sel (1,0). Kemudian sel (1,0) menjadi sel sekarang, dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*.

y/x	0	1	2	3	4
0	13	7	15	15	15
1	15	15	15	15	15
2	15	15	15	15	15
3	15	15	15	15	15
4	15	15	15	15	15

Iterasi 2

Sel sekarang (1,0). Dipilih sel tetangga secara acak yaitu (1,1), dan didapat arah = $(1,1) - (1,0) = (0,1)$ yaitu Selatan. Ubin sel sekarang (1,0) menjadi ubin *id* 3 dan ubin sel selanjutnya (1,1) menjadi ubin *id* 14. Kemudian sel (1,1) menjadi sel sekarang, dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*.

y/x	0	1	2	3	4
0	13	3	15	15	15
1	15	14	15	15	15
2	15	15	15	15	15
3	15	15	15	15	15
4	15	15	15	15	15

Iterasi 3

Sel sekarang (1,1). Dipilih sel tetangga secara acak yaitu (2,1), dan didapat arah = $(2,1) - (1,1) = (1,0)$ yaitu Timur. Ubin sel sekarang (1,1)

menjadi ubin *id* 12 dan ubin sel selanjutnya (2,1) menjadi ubin *id* 7. Kemudian sel (2,1) menjadi sel sekarang, dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*.

y/x	0	1	2	3	4
0	13	3	15	15	15
1	15	12	7	15	15
2	15	15	15	15	15
3	15	15	15	15	15
4	15	15	15	15	15

Iterasi 4

Sel sekarang (2,1). Dipilih sel tetangga secara acak yaitu (2,2), dan didapat arah = (2,2) - (2,1) = (0,1) yaitu Selatan. Ubin sel sekarang (2,1) menjadi ubin *id* 3 dan ubin sel selanjutnya (2,2) menjadi ubin *id* 14. Kemudian sel (2,2) menjadi sel sekarang, dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*.

y/x	0	1	2	3	4
0	13	3	15	15	15
1	15	12	3	15	15
2	15	15	14	15	15
3	15	15	15	15	15
4	15	15	15	15	15

Iterasi 10

Sel sekarang (3,0). Dipilih sel tetangga secara acak yaitu (2,0), dan didapat arah = $(2,0) - (3,0) = (-1,0)$ yaitu Barat. Ubin sel sekarang (3,0) menjadi ubin *id* 5 dan ubin sel selanjutnya (2,0) menjadi ubin *id* 13. Kemudian sel (2,0) menjadi sel sekarang, dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*.

y/x	0	1	2	3	4
0	13	3	13	5	3
1	15	12	3	9	6
2	15	15	12	6	15
3	15	15	15	15	15
4	15	15	15	15	15

Pada iterasi ke-10 sel sekarang (2,0) tidak memiliki tetangga yang bisa dipilih, maka algoritma akan mengeluarkan sel dari *stack* satu per satu sampai ditemukan sel yang mempunyai tetangga yang bisa dipilih.

y/x	0	1	2	3	4
0	13	3	13	5	3
1	15	12	3	9	6
2	15	15	12	6	15
3	15	15	15	15	15
4	15	15	15	15	15

Iterasi 11

Pada iterasi ke-10 ditemukan sel sekarang (4,1). Dipilih sel tetangga secara acak yaitu (4,2), dan didapat arah = (4,2) - (4,1) = (0,1) yaitu Selatan. Ubin sel sekarang (4,1) menjadi ubin *id* 2 dan ubin sel selanjutnya (4,2) menjadi ubin *id* 14. Kemudian sel (4,2) menjadi sel sekarang, dimasukkan ke dalam *stack* dan dikeluarkan dari *unvisited*.

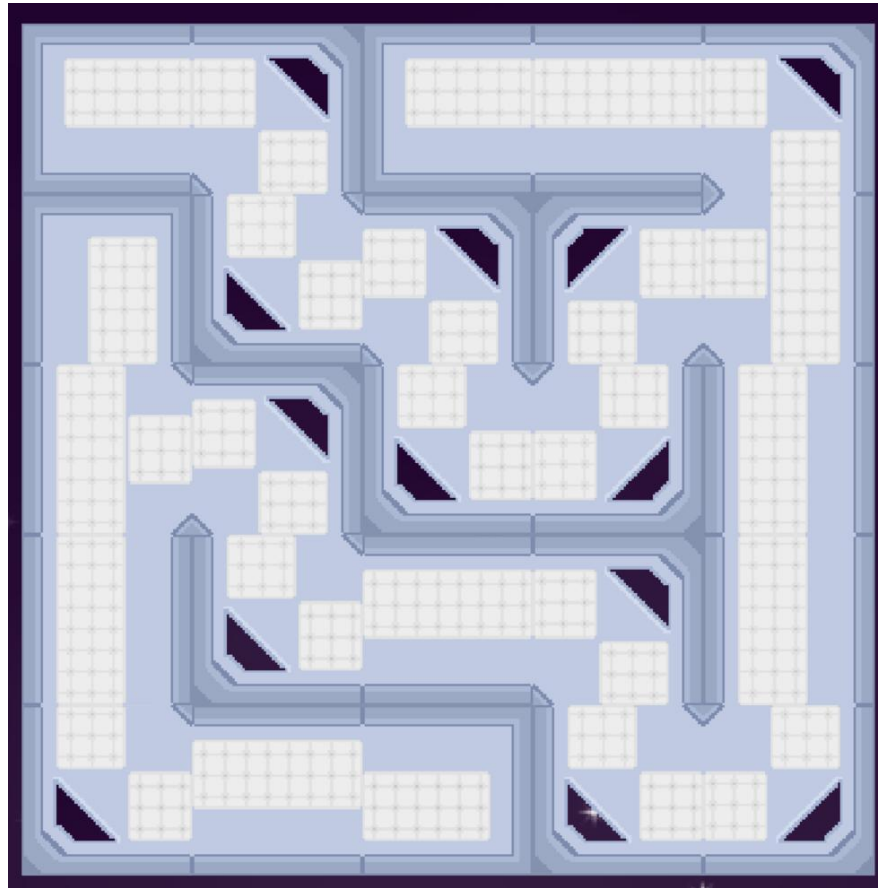
y/x	0	1	2	3	4
0	13	3	13	5	3
1	15	12	3	9	2
2	15	15	12	6	14
3	15	15	15	15	15
4	15	15	15	15	15

Iterasi 24

Sel sekarang (0,2). Dipilih sel tetangga secara acak yaitu (0,1), dan didapat arah = $(0,1) - (0,2) = (0,-1)$ yaitu Utara. Ubin sel sekarang (0,2) menjadi ubin *id* 8 dan ubin sel selanjutnya (0,1) menjadi ubin *id* 11. Karena tidak ada lagi sel di dalam *unvisited*, maka iterasi diakhiri dan didapat bentuk labirin sebagai berikut.

y/x	0	1	2	3	4
0	13	3	13	5	3
1	11	12	3	9	2
2	8	3	12	6	10
3	10	12	5	3	10
4	12	5	7	12	6

Jika digunakan ubin yang mempunyai *id* sesuai maka hasil labirin menjadi.



Gambar 4. 8 Hasil Pengujian Algoritma DFS pada Editor Godot

4.3. Hasil Pengujian

Tahap ini bertujuan untuk mengevaluasi hasil implementasi sistem agar sesuai dengan fungsi-fungsi yang telah ditentukan pada tahap analisis dan perancangan sistem. Pengujian sistem dilakukan pada penggenerasi labirin dalam permainan. Dalam penelitian ini, sistem diuji mulai permainan, evaluasi labirin yang digenerasi, dan kompleksitas algoritma *running time*.

4.3.1. Pengujian mulai permainan

Tabel 4. 2 Pengujian Mulai Permainan

Kelas Uji	Skenario Uji	Hasil yang Diharapkan	Kesimpulan
Mulai Permainan	Memasuki <i>Level</i>	Menggenerasi Labirin	[✓] Berhasil [] Gagal
		Menggenerasi Karakter	
		Menggenerasi Perangkap	
		Menggenerasi <i>item</i> pembantu pemain	

4.3.2. Pengujian evaluasi labirin

Labirin dalam suatu *level* bisa dievaluasi dengan mengumpulkan data skor pemain yang pada program penelitian ini berupa skor Medali. Setiap medali dapat diartikan sesuai dengan perkiraan waktu, dengan Medali Emas diberikan jika selesai dalam 0-30 detik, Medali Perak dalam 31-50 detik, dan Medali Perunggu dalam waktu lebih dari 50 detik. Pada pengujian ini data dikumpulkan dari 5 pemain yang memainkan permainan hasil penelitian ini.

Tabel 4. 3 Pengujian Evaluasi Labirin

Pemain	Medali Emas	Medali Perak	Medali Perunggu
1	7	2	4
2	25	11	4
3	0	0	1

4	7	5	4
5	-	-	2

Dari tabel 4.3, dapat disimpulkan bahwa dalam rata-rata tingkat kesulitan berhubungan dengan frekuensi pemain memainkan suatu *level*, karena semakin banyak pemain memainkan suatu level, semakin mudah bagi pemain untuk menyelesaikan labirin.

4.3.3. Kompleksitas Algoritma

Kalkulasi kompleksitas waktu algoritma dapat dilakukan dengan analisa kode program yang diterapkan menggunakan notasi *Big-Θ*. Hasil yang didapat dijelaskan pada tabel 4.4.

Dari tabel , dapat disimpulkan bahwa tingkat kesulitan berhubungan dengan frekuensi pemain memainkan suatu *level*, karena semakin banyak pemain memainkan suatu level, semakin mudah bagi pemain untuk menyelesaikan labirin.

Tabel 4. 4 Kompleksitas Algoritma Depth-First Search

No	Kode Program	C	#	C#
1.	func make_maze():	C ₁	1	C ₁
2.	var unvisited = []	C ₂	1	C ₂
3.	var stack = []	C ₂	1	C ₂
4.	Map.clear()	C ₃	1	C ₃

5.	for x in range(width):	C ₄	n	C _{4n}
6.	for y in range(height):	C ₄	m	C _{4m}
7.	unvisited.append(Vector2(x, y))	C ₅	n*m	C _{5nm}
8.	Map.set_cellv(Vector2(x, y), N E S W)	C ₅	n*m	C _{5nm}
9.	var current = Vector2(0, 0)	C ₂	1	C ₂
10.	unvisited.erase(current)	C ₆	1	C ₆
11.	while unvisited:	C ₇	n	C _{7n}
12.	var neighbors = check_neighbors(current, unvisited)	C ₂	n	C _{2n}
13.	if neighbors.size() > 0:	C ₈	n	C _{8n}
14.	var next = neighbors[randi() % neighbors.size()]	C ₂	n	C _{2n}
15.	stack.append(current)	C ₅	n	C _{5n}
16.	var dir = next - current	C ₂	n	C _{2n}
17.	var current_walls = Map.get_cellv(current) - cell_walls[dir]	C ₂	n	C _{2n}
18.	var next_walls = Map.get_cellv(next) - cell_walls[-dir]	C ₂	n	C _{2n}
19.	Map.set_cellv(current, current_walls)	C ₅	n	C _{5n}
20.	Map.set_cellv(next, next_walls)	C ₅	n	C _{5n}
21.	current = next	C ₂	n	C _{2n}
22.	unvisited.erase(current)	C ₆	n	C _{6n}
23.	elif stack:	C ₈	n	C _{8n}
24.	current = stack.pop_back()	C ₂	n	C _{2n}

Perhitungan kompleksitas yang didapat dari tabel 4.3 adalah:

$$T(n) = C_1 + 3C_2 + C_3 + C_6 + 7C_2n + C_4n + 3C_5n + C_6n + C_7n + 2C_8n + C_4m + 2C_5nm$$

$$T(n) = C_1 + 3C_2 + C_3 + C_6 + (7C_2 + C_4 + 3C_5 + C_6 + C_7 + 2C_8)n + (C_4)m + (2C_5)nm$$

$$T(n) = \Theta(nm)$$

Disimpulkan kompleksitas algoritma Depth-First Search adalah $\Theta(n*m)$.

BAB 5

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan analisis, perancangan, dan implementasi algoritma *Depth-First Search* pada *video game* labirin, didapat beberapa kesimpulan sebagai berikut:

1. Implementasi algoritma *Depth-First Search* untuk pembuatan labirin 2 dimensi berhasil diimplementasikan dengan baik dengan waktu rata-rata 1,2499 ms dan mempunyai kompleksitas algoritma $O(nm)$.
2. Objek-objek dalam permainan seperti *item* pembantu dapat bekerja dengan baik dan perangkat dapat digenerasi secara *random* sehingga membuat tingkat kesulitan bagi pemain.
3. Permainan dapat dimainkan di OS *Windows* tanpa koneksi internet.

5.2 Saran

Beberapa saran yang ingin disampaikan penulis berdasarkan pengalaman yang diperoleh penulis dalam pengerjaan penelitian ini adalah:

1. Mengembangkan permainan yang bisa dijalankan di beberapa sistem operasi populer lainnya (macOS, Android, Ios, Linux).
2. Menyediakan alur *progression* permainan yang lebih padat dan tepat.
3. Menyediakan elemen-elemen pada permainan yang meningkatkan ketertarikan pemain
4. Mengembangkan permainan yang mempunyai lebih banyak konten dari permainan sebelumnya.
5. Menyampaikan penelitian dengan lebih akurat sehingga tidak terkesan ambigu

DAFTAR PUSTAKA

- Botea, A., Bouzy, B., Buro, M., Bauckhage, C., & Nau, D. (2013). Pathfinding in Games. In S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, & J. Togelius (Eds.), *Artificial and Computational Intelligence in Games* (Vol. 6, pp. 21–31). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. <https://doi.org/10.4230/DFU.Vol6.12191.21>
- Brodie, I. (2002). *The Medium of the Video Game*. By Mark J. P. Wolf, ed. Foreword by Ralph H. Baer. (Austin: University of Texas Press, 2001. xx + 203 p., ISBN 0-292-79150-X). 24, 264. <https://doi.org/10.7202/006656ar>
- Dalem, I. B. G. W. A. (2018). Penerapan algoritma A*(Star) menggunakan graph untuk menghitung jarak terpendek. *Jurnal RESISTOR (Rekayasa Sistem Komputer)*, 1(1), 41–47.
- Hendrawan, Y. F. (2018). A Maze Game on Android Using Growing Tree Method. *Journal of Physics: Conference Series*, 953, 12148. <https://doi.org/10.1088/1742-6596/953/1/012148>
- Lawson, V. I. (2019). *Implementasi Metode Heuristic Dalam Game ELLER'S MAZE*. Universitas Sumatera Utara.
- Ricky, R. (2021). *Rancang Bangun Game Platformer Bintang Kecil Menggunakan Godot Engine*. Universitas Putera Batam.
- L'Ecuyer, P. (2012). *Random number generation*. Springer.
- Nugraha, D. W. (2012). Penerapan Kompleksitas Waktu Algoritma Prim untuk Menghitung Kemampuan Komputer dalam Melaksanakan Perintah. *Foristek*, 2(2).
- Zhang, H., Zeng, X., & Duan, L. (2018, 2018/12//). Procedural Content Generation for Room Maze. *Proceedings of the 2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*,
- Holzmann, G. J., Peled, D. A., & Yannakakis, M. (1996). On nested depth first search. *The Spin Verification System*, 32, 23-32.

- Lumenta, A. S. M. (2014). Perbandingan Metode Pencarian Depth-First Search, Breadth-First Search Dan Best-First Search Pada Permainan 8-Puzzle. *Jurnal Teknik Elektro dan Komputer*, 3(1), 11-16.
- Iyanda, J. (2018). A Comparative Analysis of Breadth First Search (BFS) and Depth First Search (DFS) Algorithms.

LISTING PROGRAM

Generasi Labirin Algoritma *Depth-First Search*:

```
func make_maze():
    var unvisited = []
    var stack = []
    Map.clear()
    for x in range(width):
        for y in range(height):
            unvisited.append(Vector2(x, y))
            Map.set_cellv(Vector2(x, y), N|E|S|W)
    var current = Vector2(0, 0)
    unvisited.erase(current)
    while unvisited:
        var neighbors = check_neighbors(current, unvisited)
        if neighbors.size() > 0:
            var next = neighbors[randi() % neighbors.size()]
            stack.append(current)
            var dir = next - current
            var current_walls = Map.get_cellv(current) - cell_walls[dir]
            var next_walls = Map.get_cellv(next) - cell_walls[-dir]
            Map.set_cellv(current, current_walls)
            Map.set_cellv(next, next_walls)
            current = next
            unvisited.erase(current)
        elif stack:
            current = stack.pop_back()
```