

**IMPLEMENTASI PENDEKATAN *HYBRID* UNTUK *PROCEDURAL*  
*CONTENT GENERATION* PEMBUATAN LEVEL *GAME 2D PLATFORMER*  
MENGUNAKAN *UNITY***

**ERICK YUDHA PRATAMA SUKKU**

**201401046**



**PROGRAM STUDI S1 ILMU KOMPUTER**

**FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI**

**UNIVERSITAS SUMATERA UTARA**

**MEDAN**

**2024**

**UNIVERSITAS SUMATERA UTARA**

**IMPLEMENTASI PENDEKATAN *HYBRID* UNTUK *PROCEDURAL*  
*CONTENT GENERATION* PEMBUATAN LEVEL *GAME 2D PLATFORMER*  
MENGUNAKAN *UNITY***

**SKRIPSI**

Diajukan untuk melengkapi tugas dan memenuhi syarat memperoleh ijazah  
Sarjana Ilmu Komputer

**ERICK YUDHA PRATAMA SUKKU**

**201401046**



**PROGRAM STUDI S1 ILMU KOMPUTER**

**FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI**

**UNIVERSITAS SUMATERA UTARA**

**MEDAN**

**2024**

**UNIVERSITAS SUMATERA UTARA**

**PERSETUJUAN**

Judul : IMPLEMENTASI PENDEKATAN HYBRID UNTUK  
PROCEDURAL CONTENT GENERATION  
PEMBUATAN LEVEL GAME 2D PLATFORMER  
MENGUNAKAN UNITY

Kategori : SKRIPSI

Nama : ERICK YUDHA PRATAMA SUKKU

Nomor Induk Mahasiswa : 201401046

Program Studi : SARJANA (S-1) ILMU KOMPUTER

Fakultas : ILMU KOMPUTER DAN TEKNOLOGI  
INFORMASI UNIVERSITAS SUMATERA UTARA

Medan, 5 Maret 2024

Komisi Pembimbing:

Dosen Pembimbing II

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

Dosen Pembimbing I

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

Diketahui/Disetujui oleh Program Studi

SI Ilmu Komputer



Dr. Amalia S.T., M.T  
NIP. 197812212014042001

**PERNYATAAN****IMPLEMENTASI PENDEKATAN *HYBRID* UNTUK *PROCEDURAL*  
*CONTENT GENERATION* PEMBUATAN LEVEL *GAME 2D PLATFORMER*  
MENGUNAKAN *UNITY*****SKRIPSI**

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

Medan, 21 Januari 2024



Erick Yudha Pratama Sukku

201401046

## PENGHARGAAN

Dengan penuh rasa hormat dan apresiasi, peneliti ingin menyampaikan penghargaan atas dukungan dan kontribusi dari berbagai pihak yang telah membantu dalam penelitian ini. Skripsi berjudul *"Implementasi Pendekatan Hybrid untuk Procedural Content Generation Pembuatan Level Game 2D Platformer menggunakan Unity"* ini berhasil diselesaikan berkat kerjasama dan bimbingan dari berbagai instansi.

Terima kasih kepada:

1. Sebagai Dekan Fakultas Ilmu Komputer Universitas Sumatera Utara, Dr. Maya Silvi Lydia B.Sc., M.Sc.
2. Dr. Amalia, ST., M.T., sebagai Ketua Prodi Program Studi Ilmu Komputer.
3. Bapak Dr. Jos Timanta Tarigan S.Kom., M.Sc., selaku Ketua Laboratorium Computer Vision dan Multimedia Program Studi Ilmu Komputer, sekaligus Pembimbing I, memberikan arahan dan bimbingan yang sangat berharga.
4. Sri Melvani Hardi S.Kom., M.Kom., Pembimbing II, memberikan arahan serta bimbingan yang sangat berharga dalam perkembangan penelitian ini.
5. *Play-tester* yang telah dengan sukarela meluangkan waktu untuk membantu peneliti, baik dengan memainkan game yang dibuat, memberikan respons selama wawancara, maupun memberikan kritik, masukan, dan saran.
6. Teman-teman seperjuangan di program studi Ilmu Komputer yang telah memberikan dukungan dan bantuan yang berarti selama proses perkuliahan.

Peneliti juga mengucapkan terima kasih kepada semua pihak yang tidak dapat disebutkan satu per satu namun turut berperan dalam kelancaran penelitian ini. Semua kontribusi dan dukungan sangat berarti dalam menyelesaikan skripsi ini.

Selain itu, peneliti ingin menambahkan rasa terima kasih kepada keluarga yang senantiasa memberikan dukungan moral dan motivasi selama penelitian berlangsung. Keberhasilan ini juga tidak terlepas dari doa dan semangat yang diberikan oleh orang-orang terdekat.

Dengan rendah hati, peneliti menyampaikan permohonan maaf atas segala kekurangan dan ketidaksempurnaan yang mungkin terdapat dalam penulisan ini. Semoga hasil penelitian ini dapat memberikan kontribusi positif bagi perkembangan ilmu pengetahuan. Terima kasih atas kesempatan ini.

Medan, 21 Januari 2024

Peneliti,

A handwritten signature in dark ink, appearing to read 'Erick', with a stylized flourish underneath.

Erick Yudha Pratama Sukku

NIM 201401046

## ABSTRAK

Peningkatan popularitas genre *2D platformer* dalam industri *video game* mendorong perlunya pendekatan inovatif dalam pembuatan level. Penelitian ini mengusulkan metode pendekatan *hybrid* menggunakan *Procedural Content Generation (PCG)* di Unity. Dengan mengintegrasikan pembuatan level manual dan algoritma prosedural, penelitian ini bertujuan menciptakan game *2D platformer*, "*Lost Labyrinths: Rogue's Odyssey*". Hasil pengujian menunjukkan bahwa metode *PCG hybrid* mampu mengatasi kekurangan metode *PCG* lainnya dalam menciptakan map dengan tema artistik tertentu dan dapat mengatur tingkat kesulitan secara seimbang. Dengan demikian, pendekatan *PCG hybrid* memberikan kontribusi signifikan dalam menghadapi tantangan pembuatan level game *2D platformer*, menciptakan pengalaman bermain yang variatif, konsisten, dan menarik.

**Kata kunci:** *Procedural Content Generation, Pendekatan Hybrid, Level Game, 2D Platformer, Unity*

# IMPLEMENTATION OF A HYBRID APPROACH FOR PROCEDURAL CONTENT GENERATION IN THE CREATION OF 2D PLATFORMER GAME LEVELS USING UNITY

## ABSTRACT

The increasing popularity of the 2D platformer genre in the video game industry necessitates innovative approaches to level design. This research proposes a hybrid approach utilizing Procedural Content Generation (PCG) in Unity. By integrating manual level creation with procedural algorithms, the study aims to develop a 2D platformer game titled "Lost Labyrinths: Rogue's Odyssey." Test results indicate that the PCG hybrid method effectively addresses shortcomings present in other PCG approaches, creating maps with specific artistic themes and balancing difficulty levels. Thus, the PCG hybrid approach significantly contributes to overcoming challenges in 2D platformer level design, delivering a gaming experience that is diverse, consistent, and appealing.

**Keywords:** *Procedural Content Generation, Hybrid Approach, Game Level, 2D Platformer, Unity, Lost Labyrinths: Rogue's Odyssey.*



## DAFTAR ISI

<b>PERSETUJUAN .....</b>	<b>ii</b>
<b>PERNYATAAN.....</b>	<b>iii</b>
<b>PENGHARGAAN.....</b>	<b>iv</b>
<b>ABSTRAK .....</b>	<b>vi</b>
<b>ABSTRACT .....</b>	<b>vii</b>
<b>DAFTAR ISI.....</b>	<b>viii</b>
<b>DAFTAR GAMBAR.....</b>	<b>x</b>
<b>DAFTAR TABEL .....</b>	<b>xii</b>
<b>DAFTAR LAMPIRAN .....</b>	<b>xiii</b>
<b>DAFTAR RUMUS .....</b>	<b>xiv</b>

<b>BAB I PENDAHULUAN.....</b>	<b>1</b>
1.1. Latar Belakang .....	1
1.2. Rumusan Masalah.....	3
1.3. Tujuan Penelitian .....	3
1.4. Manfaat Penelitian .....	3
1.5. Batasan Masalah .....	4
1.6. Penelitian Relevan .....	4
<b>BAB II DAFTAR PUSTAKA .....</b>	<b>6</b>
2.1. <i>Game Platformer</i> .....	6
2.2. <i>Procedural Content Generation</i> .....	7
2.3. <i>Unity</i> .....	8
<b>BAB III ANALISIS DAN PERANCANGAN SISTEM.....</b>	<b>10</b>
3.1. Metodologi Penelitian.....	10
3.2. Analisis dan Perancangan .....	11
3.2.1. Analisis Masalah.....	11
3.2.2. Analisis Kebutuhan.....	11
3.3. Perancangan Cerita Permainan .....	12
3.4. Perancangan Arsitektur Sistem .....	12
3.4.1. Pembuatan <i>Template</i> Manual Ruangan Level .....	13

3.4.2.	Pembuatan Struktur Level .....	18
3.4.3.	<i>Level Generator</i> .....	20
3.4.4.	<i>Obstacle Generator</i> .....	22
3.4.5.	<i>Enemy Generator</i> .....	23
3.4.6.	<i>Item Generator</i> .....	25
3.4.7.	<i>Player Spawn</i> .....	26
3.5.	Desain <i>User Interface</i> .....	26
3.6.	Rencana Pengujian .....	27
3.6.1.	Analisis Tingkat Kesulitan Level .....	27
3.6.2.	Pengujian Level Oleh <i>Playtester</i> .....	28
<b>BAB IV IMPLEMENTASI DAN PENGUJIAN SISTEM .....</b>		<b>29</b>
4.1.	Video Game “Lost Labyrinths: Rogue’s Odyssey” .....	29
4.2.	Pengujian Program .....	33
4.2.1.	Analisis Tingkat Kesulitan Level .....	33
4.2.2.	Hasil Pengujian Oleh Sampel .....	40
<b>BAB V KESIMPULAN DAN SARAN .....</b>		<b>44</b>
5.1.	Kesimpulan .....	44
5.2.	Saran .....	44
<b>DAFTAR PUSTAKA .....</b>		<b>45</b>
<b>LAMPIRAN .....</b>		<b>47</b>

## DAFTAR GAMBAR

<b>Gambar 1.1</b> Visualisasi <i>PCG Level Spelunky</i> .....	2
<b>Gambar 1.2</b> Graf Struktur Level <i>Dead Cells</i> .....	2
<b>Gambar 2.1</b> Contoh Game <i>Platformer, Celeste</i> (2018) .....	6
<b>Gambar 2.2</b> <i>PCG Dunia Game Minecraft Menggunakan Perlin Noise</i> .....	8
<b>Gambar 2.3</b> Logo <i>Unity</i> .....	8
<b>Gambar 2.4</b> Tampilan <i>Unity</i> .....	9
<b>Gambar 3.1</b> Metode <i>Waterfall</i> .....	10
<b>Gambar 3.2</b> Rancangan <i>Flow Sistem</i> .....	13
<b>Gambar 3.3</b> Konsep Visual Level Tema " <i>Cave</i> " Sumber: <i>The Bardent Asset Pack – Itch.io</i> .....	14
<b>Gambar 3.4</b> Konsep Data <i>Room</i> .....	14
<b>Gambar 3.5</b> Contoh Prefab Ruangan .....	17
<b>Gambar 3.6</b> Hasil Pembuatan Struktur Level .....	22
<b>Gambar 3.7</b> Contoh Prefab Ruangan Dengan <i>Spike Tiles</i> .....	23
<b>Gambar 3.8</b> Hasil <i>Spawn Spike Prefab</i> Sebelumnya .....	23
<b>Gambar 3.9</b> Contoh Musuh <i>Boss</i> (Indikator Tengkorak) .....	24
<b>Gambar 3.10</b> Contoh Prefab Ruangan dengan <i>Enemy Tiles</i> .....	25
<b>Gambar 3.11</b> Hasil Level Dengan <i>Elite Enemy</i> .....	25
<b>Gambar 3.12</b> <i>UI Minimap dan Gold Player</i> .....	27
<b>Gambar 3.13</b> <i>UI Full Map</i> .....	27
<b>Gambar 4.1</b> Tampilan <i>Main Menu Game</i> .....	29
<b>Gambar 4.2</b> Tampilan <i>Settings dan Guide Game</i> .....	29
<b>Gambar 4.3</b> Tampilan <i>Pause Game</i> .....	30
<b>Gambar 4.4</b> <i>Gameplay Action Game</i> .....	30
<b>Gambar 4.5</b> <i>Treasure Event Dalam Game</i> .....	31
<b>Gambar 4.6</b> <i>UI Treasure Game</i> .....	31
<b>Gambar 4.7</b> <i>Shop Event Dalam Game</i> .....	31
<b>Gambar 4.8</b> <i>UI Shop Game</i> .....	32
<b>Gambar 4.9</b> Level Bertema " <i>Cave</i> " .....	32
<b>Gambar 4.10</b> <i>UI Map</i> untuk Membantu Navigasi Level .....	33
<b>Gambar 4.11</b> Grafik Pengaruh <i>difficulty Terhadap stoppingRoomNum</i> .....	34
<b>Gambar 4.12</b> Grafik Uji Coba Rumus <i>Spawn Spike</i> .....	35
<b>Gambar 4.13</b> Pengaruh <i>treasureSpawnCoefficient</i> Pada Jumlah <i>Treasure</i> .....	38
<b>Gambar 4.14</b> Level " <i>Cave</i> " <i>PCG Perlin Noise</i> (a), Sumber: Ethan Bruins – <i>Unity Blog</i> .....	40
<b>Gambar 4.15</b> Level " <i>Cave</i> " <i>PCG Random Walk</i> (b), Sumber: Ethan Bruins – <i>Unity Blog</i> .....	40

<b>Gambar 4.16</b> Level “Cave” PCG Directional Tunnel (c), Sumber: Ethan Bruins – <i>Unity Blog</i> .....	41
<b>Gambar 4.17</b> Level “Cave” PCG Cellular Automata (d), Sumber: Ethan Bruins – <i>Unity Blog</i> .....	41
<b>Gambar 4.18</b> Level “Cave” PCG Pendekatan Hybrid (e).....	41

## DAFTAR TABEL

<b>Tabel 3.1</b> <i>Tilemap Layer</i> .....	16
<b>Tabel 3.2</b> Tipe dan Aturan Ruangan .....	17
<b>Tabel 3.3</b> Variabel Penempatan Posisi Indikator .....	21
<b>Tabel 3.4</b> <i>Multiplier Rarity Item</i> .....	26
<b>Tabel 4.1</b> Hasil Uji Coba Rumus <i>Spawn Spike</i> .....	35
<b>Tabel 4.2</b> <i>Outcome</i> Probabilitas <i>Spawn</i> Musuh .....	36
<b>Tabel 4.3</b> Hasil Uji Coba Rumus <i>Spawn Enemy</i> .....	37
<b>Tabel 4.4</b> <i>Multiplier Rarity Item</i> .....	39
<b>Tabel 4.5</b> Hasil Uji Coba Rumus <i>Spawn Item</i> .....	39
<b>Tabel 4.6</b> Hasil Kuisioner Perbandingan Metode <i>PCG</i> .....	42
<b>Tabel 4.7</b> Hasil Kuisioner Aspek Game .....	42

**DAFTAR LAMPIRAN**

<b>Lampiran 1</b> KUISIONER PERBANDINGAN METODE PROCEDURAL CONTENT GENERATION DAN PENGALAMAN BERMAIN GAME “LOST LABYRINTHS: ROGUE’S ODYSSEY” .....	47
<b>Lampiran 2</b> TABULASI JAWABAN RESPONDEN .....	48

**DAFTAR RUMUS**

Rumus (1) Jumlah Ruangan Level.....	33
Rumus (2) Kemungkinan Duri.....	34
Rumus (3) Kemungkinan Musuh.....	36
Rumus (4) Jumlah <i>Treasure</i> Level .....	37
Rumus (5) Kemungkinan <i>Rarity</i> Item.....	39

## BAB I

### PENDAHULUAN

#### 1.1. Latar Belakang

Permainan 2D *platformer* merupakan salah satu genre *video game* paling populer saat ini, terutama di lingkungan komunitas *indie game*. Beberapa game 2D *platformer* yang paling populer saat ini yaitu *Dead Cells*, *Hollow Knight*, *Spelunky*, dan masih banyak lagi. Adapun kesamaan dari banyak *game* di genre ini yaitu penggunaan *Procedural Content Generation (PCG)* dalam pembuatan level *game*.

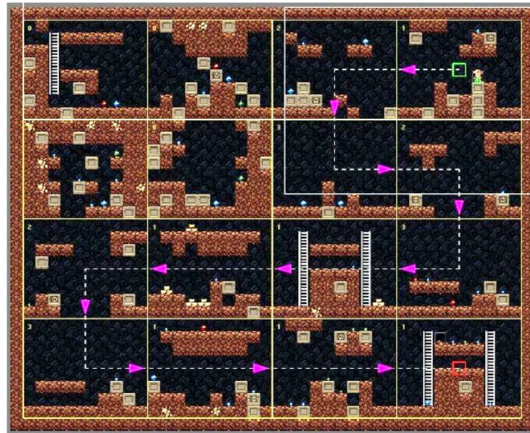
*Procedural Content Generation (PCG)* adalah metode menggunakan prosedur dalam memberikan instruksi kepada komputer untuk membuat konten dengan berbagai cara (Watkins 2016). Implementasi *PCG* saat ini semakin banyak digunakan karena keuntungan yang ditawarkannya dalam hal mempercepat proses pembuatan konten, mengurangi anggaran pengembangan, dan memfasilitasi pembuatan variasi konten yang tak terbatas (Shaker dkk., 2015). Akan tetapi metode *PCG* sendiri tidaklah sempurna.

Sebagian besar teknik yang digunakan sejauh ini untuk pembuatan level mengalami masalah kurangnya kendali dalam desain, karena biasanya sulit untuk menentukan batasan atau *requirements* desain dan cenderung menghasilkan struktur yang terasa tidak alami (Shaker dkk., 2015). Untuk mengatasi masalah ini, digunakan metode pendekatan *hybrid*, menggabungkan metode pembuatan tangan dan algoritma prosedural untuk membuat peta yang bervariasi dan tetap memiliki konsistensi dan perasaan alami.

Salah satu permainan komersial terkenal yang menggunakan pendekatan ini untuk membuat peta permainan 2D *platformer* adalah *Spelunky* (2013). Permainan ini berhasil menggunakan teknik *PCG* untuk menghasilkan variasi struktur yang unik setiap kali dimainkan dan template buatan tangan untuk mengendalikan struktur level (Baghdadi dkk., 2015). Pendekatan dalam pembuatan level *Spelunky* diusulkan oleh Baghdadi (2015) menggunakan algoritma genetika untuk menghasilkan level baru

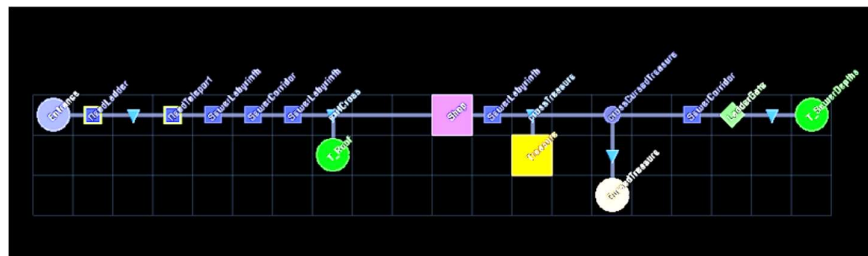


sesuai dengan *requirements* estetika dan desain. Sebuah graf digunakan sebagai representasi genetik dalam proses evolusi untuk menggambarkan struktur level dan hubungan antara ruangan, sementara *agent-based method* digunakan untuk menentukan desain interior ruangan. Hasilnya menunjukkan bahwa variasi konten bermain yang tak terbatas yang memenuhi persyaratan kesulitan yang telah ditentukan dapat dihasilkan dengan efisien.



**Gambar 1.1** Visualisasi *PCG* Level *Spelunky*

Permainan lain yang menggunakan metode serupa dalam pembuatan peta 2D platformer adalah *Dead Cells* (2018). Berdasarkan Sebastien Benard (2017), *Lead Designer* dari *Dead Cells*, permainan ini menggunakan pendekatan *hybrid* antara *procedural generation* dan *handmade level* untuk memberikan level perasaan konsistensi dan variasi yang banyak. Struktur level dibuat menggunakan graf dan level dibuat dengan menggabungkan *template* ruangan yang dibuat tangan berdasarkan graf struktur level. Terakhir, jumlah monster dalam level diatur dan ditempatkan sesuai dengan kriteria tertentu, menghasilkan level yang kaya variasi.



**Gambar 1.2** Graf Struktur Level *Dead Cells*

Dalam penelitian ini, pendekatan *hybrid* digunakan dalam pembuatan level

sebuah *game 2D platformer* secara prosedural. Pembuatan *template* ruangan dibuat *handmade* menggabungkan aspek *spawning obstacle* dari *Spelunky* dan *spawning item* dan musuh dari *Dead Cells*. Parameter *spawning obstacle*, *item*, dan musuh di level dikalkulasi berdasarkan tingkat kesulitan yang ditentukan. Struktur level direpresentasikan dalam bentuk graf dan dibuat berdasarkan implementasi dari *Dead Cells*. *Template* ruangan digabungkan secara prosedural untuk menghasilkan satu level permainan yang bervariasi namun tetap terasa konsisten dan alami.

### 1.2. Rumusan Masalah

Pembuatan level *game* menggunakan *Procedural Content Generation (PCG)* sepenuhnya memiliki beberapa kelemahan, seperti penurunan kualitas desain level serta kurangnya perasaan konsistensi dan kesesuaian estetika visual dalam level dengan tema tertentu pada *game*. Oleh karena itu diperlukan suatu pendekatan dalam *PCG* untuk mengatasi kelemahan tersebut.

### 1.3. Tujuan Penelitian

Tujuan dari penelitian ini yaitu membangun sebuah *game 2D platformer* menggunakan *Procedural Content Generation (PCG)* pendekatan *hybrid* dalam pembuatan level di *Unity*. Menggabungkan pembuatan level *handmade* dan algoritma prosedural bertujuan untuk menciptakan level *game 2D platformer* yang memiliki variasi, konsistensi tema, dan perasaan alami.

### 1.4. Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai berikut.

1. Membantu pengembang *game 2D platformer* untuk menciptakan level dengan kombinasi variasi yang menarik dan tetap terasa konsisten.
2. Membantu pengembang *game* dalam meningkatkan efisiensi pengembangan level, mengurangi anggaran pengembangan, dan mempercepat proses pembuatan konten dalam *game*.
3. Menjadi kontribusi dalam bidang *Procedural Content Generation* dengan menggabungkan elemen-elemen kreatif buatan tangan manusia dengan keunggulan algoritma prosedural, membuka jalan untuk penelitian lebih lanjut dalam pengembangan metode *PCG*.

### 1.5. Batasan Masalah

Berikut batasan masalah dalam penelitian ini.

1. *Game* dibuat memiliki genre *2D platformer*
2. *Game* dibuat menggunakan *game engine Unity*
3. Algoritma ditulis menggunakan bahasa pemrograman *C#* untuk *Unity*.
4. Template ruangan dalam level yang digabungkan menjadi sebuah level dibuat secara *handmade* menggunakan fitur *Grid* dan *Tilemaps* di *Unity*.
5. Generator struktur level, generator *obstacle*, generator musuh, dan generator *item* dibuat menggunakan algoritma prosedural.
6. *Output* generator dipengaruhi oleh tingkat kesulitan level dan generator dibatasi aturan-aturan tertentu agar tidak terlalu *random*.

### 1.6. Penelitian Relevan

- 1) Penelitian oleh Baghdadi, dkk. (2015) berjudul “*A Procedural Method for Automatic Generation of Spelunky Levels*”. Dalam penelitian ini, penulis mengusulkan pendekatan berbasis pencarian evolusioner untuk menghasilkan level secara otomatis dalam permainan *Spelunky* yang menggabungkan genre *2D platformer* dan *rogue-like*. Algoritma genetika digunakan untuk menciptakan level baru sesuai dengan persyaratan estetika dan desain, menggunakan representasi graf untuk struktur level dan metode berbasis agen untuk desain interior ruangan. Hasil penelitian menunjukkan bahwa variasi konten yang tak terbatas yang memenuhi persyaratan kesulitan dapat dihasilkan secara efisien.
- 2) Penelitian oleh Ripamonti dkk. (2017) berjudul “*Procedural content generation for platformers: designing and testing FUN PLEdGE*”. Penelitian ini mengatasi tantangan dalam pengembangan *video game* dengan fokus pada peran penting desainer level dalam menciptakan pengalaman bermain yang menyenangkan sambil mematuhi berbagai aturan. Peneliti menciptakan FUN PLEdGE, sebuah alat pembuatan level otomatis yang memberikan kebebasan kepada desainer level untuk memodifikasi level secara manual, sambil memberikan arahan untuk meningkatkan pengalaman bermain. Hasilnya

adalah alat yang dapat mempercepat pengembangan permainan sambil mempertahankan kualitas dan kebebasan dalam desain level.

- 3) Penelitian oleh Gellel dan Sweetser (2020) berjudul “*A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels*”. Penelitian ini mengusulkan pendekatan dalam pembuatan level untuk permainan *dungeon-style roguelike* menggunakan metode *PCG*. Peneliti menggabungkan teknik *hybrid* yang menggunakan *context-free grammar* untuk menghasilkan deskripsi level dan proses yang terinspirasi dari algoritma *cellular automata* untuk menghasilkan ruang fisik. Hasilnya adalah generator level yang berhasil menghasilkan struktur *dungeon* yang selalu dapat dimainkan dengan variasi yang memadai. Penelitian ini menunjukkan nilai besar dalam pendekatan *hybrid* untuk desain level otomatis.

## BAB II

### TINJAUAN PUSTAKA

#### 2.1. *Game Platformer*

Definisi modern dari *game platformer* adalah sebuah *game* yang mekanik utamanya melibatkan melompati atau melewati rintangan (Bycer, 2023). Genre permainan ini berfokus pada navigasi karakter pemain melalui berbagai level yang diisi dengan *platform*, rintangan, dan musuh. Genre *game platformer* ini menjadi salah satu genre *game* populer saat ini dengan pengembang *indie game* yang merilis *game* seperti *Hollow Knight* (2017), *Dead Cells* (2018), dan *Celeste* (2018).



**Gambar 2.1** Contoh *Game Platformer*, *Celeste* (2018)

*Game platformer* bisa dibagi menjadi:

a. *2D platformer*

Karakter *player* terbatas bergerak di ruang dua dimensi (kiri, kanan, atas, bawah). Pergerakan kamera mengikuti posisi *player* dari samping secara otomatis. Contohnya adalah *Celeste* (2018).

b. *3D platformer*

Karakter *player* bergerak di ruang 3 dimensi. Kamera terkunci melihat *player* dan bisa digerakkan untuk membantu navigasi level tiga dimensi. Contohnya adalah *Super Mario Odyssey* (2017).

*Game 2D platformer* dipilih untuk dibuat dalam penelitian ini karena genre ini memiliki daya tarik tersendiri dalam hal estetika, desain level yang kreatif, dan

kesederhanaan dalam navigasi karakter. Genre ini memungkinkan pengembang untuk fokus pada mekanik gameplay yang kuat dan desain level yang detail.

## 2.2. *Procedural Content Generation*

*Procedural Content Generation (PCG)* mengacu pada praktik, dalam *video game* dan permainan lainnya, untuk menghasilkan konten seperti level, misi, atau karakter menggunakan algoritma. Pembuatan metode ini dipicu oleh kebutuhan untuk membuat *game* yang memiliki *replayability*, mengurangi beban pembuatan manual, membatasi kebutuhan ruang penyimpanan, dan membuat estetika tertentu dalam *game* (Risi & Togelius, 2019).

Penggunaan *PCG* untuk menciptakan peta dan level dalam permainan komputer dimulai sejak tahun 1970-an dengan *game* seperti *Beneath Apple Manor* dan *Rogue*, dan merupakan kasus penggunaan *PCG* yang paling umum dalam *game*. Teknik-teknik *PCG* sering kali sepenuhnya otomatis, dalam arti bahwa pengguna hanya perlu melakukan konfigurasi awal, seperti menentukan parameter dan/atau batasan keluaran, sebelum konten dihasilkan (Silva dkk., 2022).

Menurut Risi dan Togelius (2019), jenis-jenis *PCG* yang dapat ditemukan dalam sebagian besar *game* yang ada saat ini disebut metode *PCG* konstruktif (*constructive PCG*). Ini berarti algoritma pembuatan konten berjalan dalam waktu yang *fixed*, tanpa iterasi, dan tidak melakukan pencarian. Untuk menghasilkan tekstur, peta ketinggian (*heightmaps*), dan konten serupa, keluarga algoritma yang sering digunakan adalah algoritma *fractal noise* seperti *Perlin noise*. Vegetasi, sistem gua, dan struktur bercabang serupa dapat dihasilkan dengan efisien menggunakan *grammar* yang diinterpretasikan secara grafis seperti *L-systems*. Metode konstruktif lain yang dipinjam dari bidang-bidang lain dalam ilmu komputer dan disesuaikan dengan kebutuhan *PCG* dalam *game* meliputi *cellular automata* dan pendekatan lain berdasarkan komputasi lokal. Metode konstruktif lainnya didasarkan pada metode yang lebih sedikit berprinsip dan lebih spesifik untuk suatu *game*. Sebagai contoh, *Spelunky* menggabungkan sejumlah potongan level yang telah dibuat sebelumnya sesuai dengan pola yang dirancang untuk memastikan jalur yang tidak terputus dari pintu masuk ke pintu keluar. Penelitian ini juga akan menggunakan metode konstruktif lain yang tidak menggunakan

prinsip algoritma khusus karena akan menggunakan pendekatan *hybrid* yang menggunakan campuran metode buatan tangan dan algoritma prosedural.



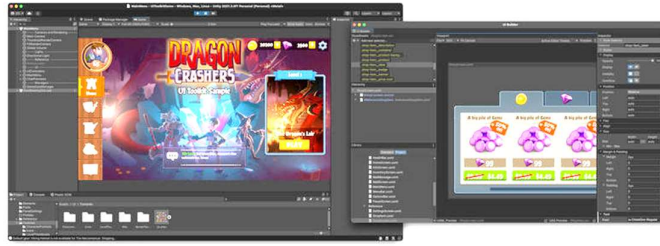
**Gambar 2.2** PCG Dunia Game *Minecraft* Menggunakan *Perlin Noise*

### 2.3. *Unity*

*Unity* adalah sebuah *cross-platform game engine* yang dikembangkan oleh *Unity Technologies* yang diluncurkan pada tahun 2005. Berdasarkan dokumentasi resmi *Unity* (2022), *game engine* ini mendukung pembuatan *game* pada mayoritas platform paling populer saat ini, seperti: *mobile*, *desktop*, *console*, *WebGL*, *VR*, *AR*, dan *XR*. Berdasarkan data yang tersedia dari layanan *hosting game* populer seperti *Steam* dan *itch.io*, *Unity* telah menjadi salah satu *engine* paling populer yang digunakan dalam pengembangan permainan dalam beberapa tahun terakhir (Shan, 2021). *Unity* dapat digunakan secara gratis, yang juga menjadikannya pilihan umum bagi mahasiswa, penghobi, seniman digital, dan banyak kelompok lainnya. Hal ini membuatnya menjadi platform yang sangat baik untuk dituju guna mendukung berbagai pengembang perangkat lunak generatif (Cook dkk., 2021).



**Gambar 2.3** Logo *Unity*



**Gambar 2.4** Tampilan *Unity*

Berikut beberapa fitur *Unity* yang menjadi pertimbangan untuk digunakan dalam penelitian ini.

*a. Physics*

*Unity* memiliki komponen bawaan yang mengatur simulasi fisika dalam permainan, termasuk deteksi dan respons terhadap peristiwa fisika seperti tabrakan, gravitasi, dan gerakan objek. Dengan sistem fisika bawaan pengembangan *game* dapat difokuskan dalam pembuatan level menggunakan *PCG*, sehingga menghemat waktu pengembangan.

*b. Grid dan Tilemaps*

*Grid* dan *Tilemap* adalah komponen penting dalam *Unity* yang digunakan dalam pengembangan *game* 2D. *Grid* adalah representasi struktur kotak beraturan yang digunakan untuk menempatkan objek dalam permainan dengan presisi. Sedangkan *Tilemap* adalah cara untuk mengatur dan menampilkan elemen grafis berulang, seperti tekstur lantai atau dinding, dalam *game* 2D. Dalam penelitian ini, penggunaan *Grid* dan *Tilemap* dalam *Unity* memberikan keuntungan signifikan karena memungkinkan penempatan elemen-elemen level secara efisien dan konsisten.

*c. Asset Store*

*Unity Asset Store* adalah platform *online* yang disediakan oleh *Unity Technologies* yang memungkinkan pengembang *game* untuk membeli, menjual, dan berbagi *game assets*, seperti model 3D, set karakter 2D, tekstur, hingga audio, yang dapat digunakan dalam pengembangan *game* dengan *Unity*. Fitur ini menghemat waktu dan tenaga dalam pembuatan *game assets* sendiri yang memungkinkan fokus lebih besar pada implementasi *PCG* dalam penelitian.

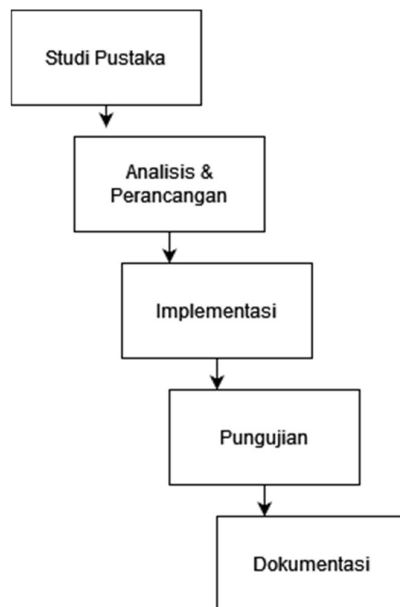


## BAB III

### ANALISIS DAN PERANCANGAN SISTEM

#### 3.1. Metodologi Penelitian

Metodologi yang dipilih untuk penelitian ini adalah metodologi pengembangan *waterfall*. Metode ini dikenal sebagai pendekatan linear yang terstruktur, di mana setiap fase pengembangan melibatkan langkah-langkah yang jelas dan terdefinisi sebelumnya. Adapun penggunaan metodologi *waterfall* dalam penelitian ini bertujuan untuk memastikan kelancaran proses pembuatan game dengan *Procedural Content Generation (PCG) Map*, serta memberikan kerangka kerja yang jelas dan terstruktur dalam pengembangan level permainan *2D platformer*.



**Gambar 3.1** Metode *Waterfall*

### 3.2. Analisis dan Perancangan

Pada tahap ini, dilakukan analisis *requirements* dan kebutuhan dalam penelitian. Selain itu dilakukan juga perancangan alur pembuatan level dengan *Procedural Content Generation (PCG)* pendekatan *hybrid*.

#### 3.2.1. Analisis Masalah

Pertama-tama, diperlukan pemahaman mendalam terhadap tantangan dan masalah yang dihadapi dalam pengembangan game *2D platformer* dengan pendekatan *Procedural Content Generation (PCG)*. Beberapa masalah utama yang muncul dari latar belakang ini adalah sebagai berikut:

##### 1) Kurangnya Kendali Desain pada *PCG*

Penggunaan metode *PCG* sering kali menghasilkan level dengan struktur yang kurang terkendali secara desain terutama apabila level memiliki tema tertentu. Keterbatasan kendali ini dapat menyebabkan kehilangan estetika dan pengalaman bermain yang diinginkan.

##### 2) Kesulitan Menentukan Batasan atau *Requirements Desain*

Proses menentukan batasan atau *requirements* desain pada *PCG* dapat menjadi rumit. Sulitnya menetapkan parameter dengan tepat dapat mengakibatkan hasil yang tidak sesuai dengan visi artistik atau tema permainan.

##### 3) Kombinasi Aspek *Handmade* dan *PCG*

Meskipun pendekatan *hybrid* dengan menggabungkan elemen *handmade* dan dapat memberikan variasi yang diinginkan, integrasi yang kurang efisien dapat mengakibatkan kompleksitas yang tidak diinginkan dan ketidaksesuaian antara level.

#### 3.2.2. Analisis Kebutuhan

Setelah mengidentifikasi masalah-masalah tersebut, selanjutnya diperlukan pemahaman terhadap kebutuhan yang harus dipenuhi agar proses pengembangan game *2D platformer* dengan *PCG* menjadi lebih efektif, antara lain:

#### 1) **Peningkatan Kendali Desain pada PCG**

Diperlukan pendekatan *PCG* yang memberikan lebih banyak kendali kepada pengembang, memungkinkan mereka menetapkan parameter desain dengan lebih tepat dan mendapatkan hasil yang sesuai dengan kebutuhan permainan.

#### 2) **Penentuan Batasan dan Requirements Desain yang Jelas**

Perlu adanya metode *PCG* yang memudahkan pengembang menentukan batasan atau *requirements* desain secara lebih jelas, sehingga level yang dihasilkan sesuai dengan tujuan artistik dan estetika permainan.

#### 3) **Optimasi Kombinasi Handmade dan PCG**

Kebutuhan untuk meningkatkan efisiensi integrasi antara elemen *handmade* dan *PCG*. Proses ini harus dapat menghasilkan level yang konsisten namun tetap memiliki variasi yang memadai untuk menjaga ketertarikan pemain.

#### 4) **Fleksibilitas dalam Penyesuaian Kesulitan**

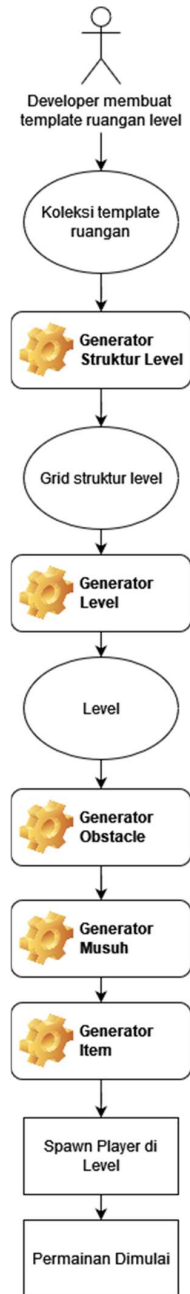
Sistem harus dapat memberikan fleksibilitas kepada pengembang untuk menyesuaikan tingkat kesulitan sesuai dengan preferensi mereka. Ini memungkinkan pengembang menciptakan pengalaman permainan yang sesuai dengan target audiens.

### 3.3. Perancangan Cerita Permainan

Pada tahap ini, cerita dibuat sebagai latar belakang permainan yang akan dibuat. Permainan memiliki latar dunia *medieval fantasy* dimana protagonis permainan adalah seorang *rogue* / pengembara yang berpetualang ke berbagai *dungeon* berbahaya dan mengalahkan berbagai musuh untuk mencari artifak legendaris yang bisa menyembuhkan adiknya dari penyakit yang tidak bisa disembuhkan.

### 3.4. Perancangan Arsitektur Sistem

Tahapan ini akan menggambarkan rencana cara kerja sistem pembuatan level game secara *PCG* pendekatan *hybrid* menggunakan *Unity*. Sistem bekerja melalui beberapa tahap dan menggunakan kerja sama beberapa komponen untuk menghasilkan sebuah level.

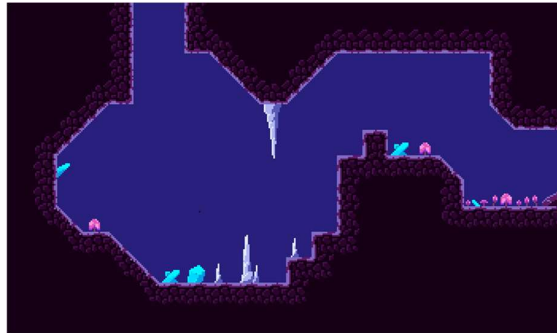


**Gambar 3.2** Rancangan *Flow* Sistem

#### 3.4.1. Pembuatan *Template* Manual Ruangan Level

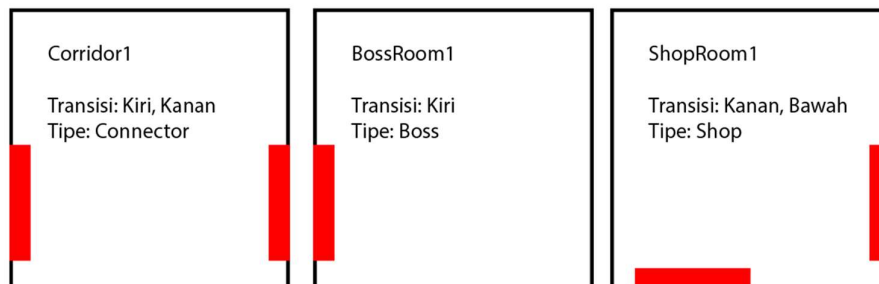
Pada tahap awal ini, seorang pengembang akan membuat *template* ruangan yang akan disatukan menjadi satu level di akhir. Pembuatan *template* secara manual bertujuan untuk membuat level yang dibuat sesuai dengan visi artistik atau tema

permainan yang diinginkan dengan menentukan visual atau *style* peletakan *tiles* dalam peta.



**Gambar 3.3** Konsep Visual Level Tema "Cave"  
Sumber: *The Bardent Asset Pack – Itch.io*

Setiap ruangan dibuat menggunakan fitur *Grid* dan *Tilemaps* di *Unity*. Sebuah ruangan didefinisikan ke dalam sebuah kelas dalam *C#* bernama *Room*.



**Gambar 3.4** Konsep Data *Room*

Adapun penerapan konsep tersebut dalam bentuk kode program adalah sebagai berikut:

```
public enum RoomType
{
    Normal,
    Start,
    Boss,
    Shop,
    Treasure
}
```

```

    }

    public class Room
    {
        public enum EntranceDirection
        {
            North,
            South,
            East,
            West
        }

        public GameObject roomPrefab;
        public RoomType roomType;
        public List<EntranceDirection> entrances;
        public Dictionary<EntranceDirection, Room>
            connectedRoomsByEntrance;
    }

```

Setiap ruangan memiliki beberapa parameter yang digunakan untuk membantu penyusunan level di proses berikutnya. Beberapa parameter tersebut diantara lain yaitu:

1) *roomType*

Tipe ruangan sebagai identifikasi untuk penyusunan level, contoh seperti: *Normal, Start, Boss, Shop, Treasure*.

2) *entrances*

*List* lokasi dimana posisi titik transisi ke ruangan lain. *List* berisi *EntranceDirection* yang merupakan arah mata angin (*North, South, West, East*).


3) *connectedRoomByEntrance*

Kumpulan pasangan data *key-value EntranceDirection* dan *Room* yang berfungsi mencatat koneksi setiap transisi ruangan ke ruangan lain. Data ini memungkinkan kelas *Room* berfungsi sebagai sebuah *graph node* untuk membantu penyusunan level.

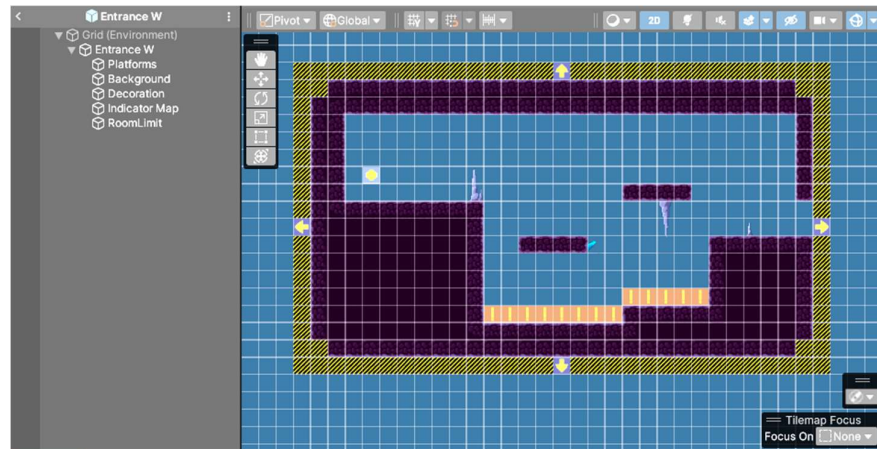
4) *roomPrefab*

Ruangan yang dibuat menggunakan *Grid* dan *Tilemap* oleh pengembang game. Setiap ruangan harus memiliki dimensi ukuran *Tile* yang sama. Di game ini setiap ruangan direncanakan memiliki ukuran 29 x 16 *Tile*. Setiap ruangan terdiri dari satu *Grid* sebagai *rule guide* peletakan setiap *Tile* dan *Tilemap* sebagai lapisan *Tile* dalam *Grid*. Setiap lapisan *Tilemap* memiliki fungsi khusus dengan detail berikut:

Tabel 3.1 *Tilemap Layer*

<i>Tilemap Layer</i>	Fungsi
<i>Platforms</i>	Lapisan <i>Tile</i> yang menjadi pijakan tempat bergerak karakter <i>Player</i> dan musuh.
<i>Background</i>	Lapisan <i>Tile</i> di belakang <i>Platforms</i> sebagai visual latar belakang level.
<i>Decoration</i>	Lapisan <i>Tile</i> di depan <i>Background</i> dan di belakang <i>Platforms</i> sebagai tempat peletakan dekorasi level.
<i>IndicatorMap</i>	<p>Berisi <i>Tile-Tile</i> indikator yang akan digunakan untuk proses khusus, seperti:</p> <ul style="list-style-type: none"> <li>a. Indikator posisi musuh </li> <li>b. Indikator posisi <i>obstacle</i> </li> <li>c. Indikator posisi <i>shop</i> </li> <li>d. Indikator posisi <i>treasure</i> </li> <li>e. Indikator posisi <i>spawn player</i> </li> <li>f. Indikator posisi <i>spawn boss</i> </li> <li>g. Indikator posisi <i>spawn exit level</i> </li> </ul>

<i>RoomLimit</i>	Lapisan <i>Tile</i> indikator Batasan ruangan untuk bantuan visual pengembang dalam membuat ruangan.
------------------	--



**Gambar 3.5** Contoh Prefab Ruangan

Posisi transisi ruangan harus diletakan di posisi yang sama untuk dan harus bersebrangan Setiap tipe ruangan memiliki aturan tertentu yang harus dipenuhi untuk membantu pembuatan level secara prosedural.

**Tabel 3.2** Tipe dan Aturan Ruangan

RoomType	Aturan
<i>Normal</i>	<ul style="list-style-type: none"> <li>- Jumlah <i>room entrance</i> <math>&gt; 0</math></li> <li>- Apabila jumlah <i>entrance</i> = 1, dikategorikan sebagai <i>normal edge room</i></li> <li>- Apabila jumlah <i>entrance</i> <math>&gt; 1</math>, dikategorikan sebagai <i>normal corridor room</i></li> <li>- Dapat memiliki <i>enemy</i> dan <i>obstacle indicator</i></li> </ul>
<i>Start</i>	<ul style="list-style-type: none"> <li>- Jumlah <i>room entrance</i> = 1</li> <li>- Memiliki 1 <i>spawn player indicator tile</i></li> <li>- Dapat memiliki <i>enemy</i> dan <i>obstacle indicator</i></li> </ul>



<i>Shop</i>	<ul style="list-style-type: none"> <li>- Jumlah <i>room entrance</i> = 2</li> <li>- Memiliki 1 <i>shop indicator tile</i></li> <li>- Ruangan akan diletakan satu ruangan sebelum <i>Boss room</i></li> <li>- Dapat memiliki <i>enemy</i> dan <i>obstacle indicator</i></li> </ul>
<i>Treasure</i>	<ul style="list-style-type: none"> <li>- Jumlah <i>room entrance</i> = 1</li> <li>- Memiliki 1 <i>treasure indicator tile</i></li> <li>- Dapat memiliki <i>enemy</i> dan <i>obstacle indicator</i></li> </ul>
<i>Boss</i>	<ul style="list-style-type: none"> <li>- Jumlah <i>room entrance</i> = 1</li> <li>- Memiliki 1 <i>treasure indicator tile</i></li> <li>- Memiliki 1 <i>exit level indicator tile</i></li> <li>- Dilarang memiliki <i>enemy</i> dan <i>obstacle indicator</i></li> </ul>

### 3.4.2. Pembuatan Struktur Level

Pembuatan struktur level menggunakan algoritma dengan bantuan graf dan *grid* serta memperhitungkan faktor kesulitan untuk menyatukan ruangan normal, ruangan *toko* (Shop), ruangan *boss* dan ruangan *harta* (Treasure). Berikut adalah penjelasan mengenai alur pembuatan struktur level:

#### a. Variabel Penting

- ***stoppingRoomNum***: Jumlah maksimum ruangan koridor yang akan dibuat berdasarkan tingkat kesulitan.
- ***currentRoomNum***: Jumlah ruangan yang sudah dibuat.

#### b. Representasi Level (*Level Grid*)

*levelGrid* adalah *Dictionary<Vector2Int, Room>*, di mana:

- *Vector2Int* adalah koordinat grid untuk setiap ruangan dalam level.
- *Room* adalah objek yang menyimpan informasi tentang jenis dan tipe ruangan, serta posisi khusus seperti pintu masuk, pintu keluar, dan lainnya.

c. **Metode Membuat Struktur Level**

- 1) Menghitung *stoppingRoomNum* dan *treasureToSpawn* berdasarkan faktor kesulitan dan koefisien yang telah ditentukan.
- 2) Inisialisasi variabel dan objek awal, termasuk ruangan awal yang merupakan jenis *Start Room*.
- 3) Menambahkan ruangan awal ke dalam *levelGrid* di posisi (0, 0).
- 4) Membuat antrian (*Queue*) untuk memproses ruangan dengan pintu masuk terbuka.
- 5) Melakukan iterasi selama antrian tidak kosong.
- 6) Untuk setiap ruangan dalam antrian:
  - a) Mengekstrak ruangan dari antrian.
  - b) Iterasi melalui setiap pintu masuk yang terbuka pada ruangan tersebut.
  - c) Jika *currentRoomNum* < *stoppingRoomNumber*, untuk setiap pintu masuk yang terbuka, letakkan ruangan baru dengan tipe *normal corridor* di posisi depan pintu tersebut.
  - d) Jika kondisi *currentRoomNum* < *stoppingRoomNumber* tidak terpenuhi, keluar dari *loop*
  - e) Menambahkan ruangan baru ke dalam antrian untuk diproses di *loop* berikutnya.

d. **Penambahan Ruangan Khusus (*Shop* dan *Treasure*):**

- 1) Setelah pembangunan level selesai, identifikasi semua pintu masuk tuangan terbuka sebagai posisi ruangan tepi (*edge room*) dan acak urutannya.
- 2) Pilih salah satu posisi ruangan tepi dan hubungkan dengan satu ruangan *shop*.
- 3) Hubungkan ruangan *boss* ke ruangan *shop* yang sudah dibuat.
- 4) Isi posisi ruangan tepi dengan ruangan *treasure* sesuai dengan jumlah yang dikalkulasikan berdasarkan kesulitan.
- 5) Untuk sisa posisi ruang tepi lain isi dengan ruangan *normal edge*.

e. **Hasil *LevelGrid* Jadi**

Mengembalikan nilai *levelGrid*, *dictionary* yang berisi posisi dan objek ruangan dalam level beserta hubungan antara mereka untuk diproses lebih lanjut.

### 3.4.3. Level Generator

Setelah struktur level jadi dalam bentuk *grid*. *Grid* level yang dibuat tersendiri hanyalah sebuah data posisi dan ruangan yang berada di posisi tersebut. *Grid* level perlu diterjemahkan menjadi nyata sebagai level keseluruhan di *game world*. Proses tersebut adalah sebagai berikut:

#### a. Validasi Level Grid

- 1) Memulai proses dengan menginisialisasi variabel *isGraphValid* sebagai *false*.
- 2) Melakukan iterasi dengan *while (!isGraphValid)*:
  - a) Membuat *levelGrid* baru
  - b) Validasi *levelGrid* jika:
    - Setidaknya ada satu ruangan dari setiap jenis.
    - Semua ruangan terhubung dengan benar.
  - c) Jika tidak valid, mengulang proses hingga mendapatkan *levelGrid* yang valid.

#### b. Spawning Room di Game World

- 1) Iterasi setiap posisi dan ruangan yang ada di *levelGrid*.
- 2) Mempersiapkan *tilemap* dan *prefab* data ruangan.
- 3) Memproses setiap *tile* dalam *tilemap prefab* dengan iterasi untuk mendapatkan posisi pemetaan dari *prefab world* ke *game world* dan jenis *tile*.
- 4) Pada setiap iterasi *tilemap*, setiap *tile* dalam *tilemap* diproses untuk menentukan posisi pemetaan dari *prefab world* ke *game world*.
- 5) Posisi *tile* dihitung berdasarkan posisi ruangan dan *offset* yang diatur pengembang untuk kebutuhan lain-lain (*posOffset*).
- 6) Iterasi dilakukan lagi dengan mengakses seluruh *tile* dalam *tilemap* menggunakan dua loop bersarang (*for (int y = 0; y < trimmedRoomSize.y; y++)* lalu *for (int x = 0; x < trimmedRoomSize.x; x++)*).
- 7) Posisi *tile* dihitung berdasarkan rumus:
 

```
int xAnchor = roomGridPos.x * trimmedRoomSize.x;
int yAnchor = roomGridPos.y * trimmedRoomSize.y;
tileSpawnPos.x = xAnchor + x + posOffset.x;
```

```
tileSpawnPos.y = yAnchor + y + posOffset.y;
```

**c. Pengecekan Tilemap “Indicator Map”**

Jika *tilemap* yang sedang diakses adalah "*Indicator Map*", maka jenis *tile* ditentukan dan posisinya disimpan sesuai dengan tipe *tile* tersebut.

**Tabel 3.3** Variabel Penempatan Posisi Indikator

Tipe Tile	Variabel Penempatan Posisi	Digunakan untuk
<i>spawnTile</i>	<i>playerSpawnPos</i>	Menandai posisi awal pemain
<i>enemyTile</i>	<i>enemyTilePos</i>	Menandai posisi musuh / <i>enemy</i>
<i>dangerTile</i>	<i>dangerTilePos</i>	Menandai posisi bahaya/ <i>spike</i>
<i>treasureTile</i>	<i>treasurePosList</i>	Menandai posisi harta karun / <i>treasure</i>
<i>shopTile</i>	<i>shopPosList</i>	Menandai posisi toko / <i>shop</i>
<i>bossTile</i>	<i>bossTilePos</i>	Menandai posisi <i>boss</i>
<i>exitTile</i>	<i>exitTilePos</i>	Menandai posisi pintu keluar ( <i>exit</i> )

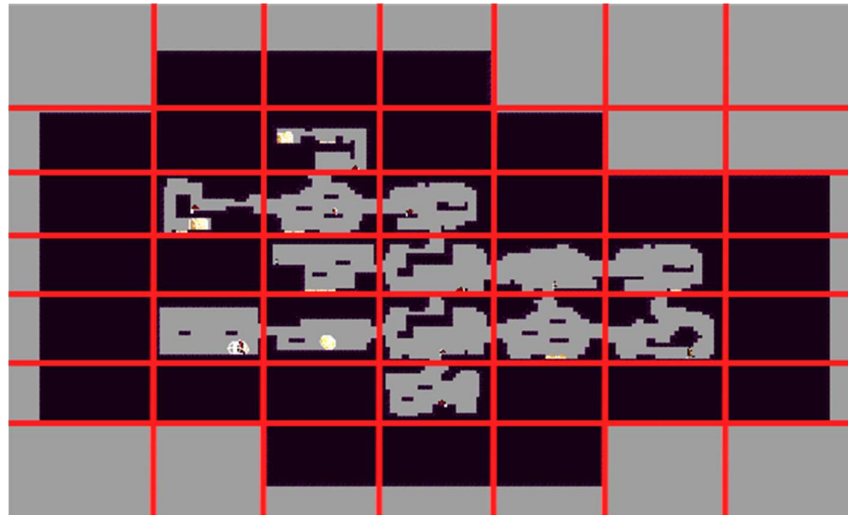
Jika *tilemap* yang diakses bukan “*Indicator Map*”, *tile* di-*spawn* ke *tilemap* yang sesuai (seperti “*Platforms*”).

**d. Mengisi Ruangan Kosong di Tepi Level**

Mengisi grid kosong di sekitar ruangan terluar dengan *default tiles*. Hal ini membantu mencegah sudut pandang pemain melihat bagian dunia yang kosong saat bermain.

#### e. Hasil Akhir

Seluruh *tilemap* di-*spawn* ke *game world* untuk setiap ruangan dalam *levelGrid*, dan posisi setiap tile ditentukan sesuai dengan jenis *tilemap* dan tipe *tile*, menghasilkan struktur level di bawah ini.



**Gambar 3.6** Hasil Pembuatan Struktur Level

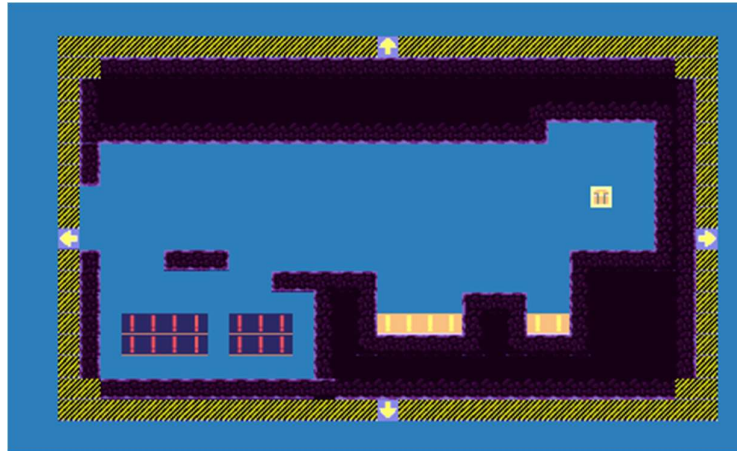
Proses ini memastikan pembuatan level yang valid dan menyediakan representasi *grid* yang digunakan untuk selanjutnya melakukan *spawning* elemen-elemen dalam permainan.

#### 3.4.4. Obstacle Generator

*Obstacle* yang akan dibuat dalam game ini adalah dalam bentuk *spike* atau duri, dimana apabila pemain bertabrakan dengan duri, karakter pemain akan terkena *damage* dan *knockback*. Adapun proses *spawning spike* dalam level yang sudah dibuat sebelumnya adalah:

- 1) Semua *spike / obstacle tile* dikelompokkan yang bersebelahan, membentuk klaster *tile*.
- 2) Hasil pengelompokan *tile* disimpan dalam *groupedSpikeTiles*. Pengelompokan dilakukan agar *spikes* muncul secara berkelompok, menciptakan area bahaya yang signifikan. Hal ini meningkatkan pengalaman bermain dengan membuat tantangan yang lebih terorganisir dan menarik.

- 3) Iterasi melalui setiap kelompok tile pada *groupedSpikeTiles*.
- 4) Hitung *spikeSpawnProbability* berdasarkan jumlah tile dalam klaster dan tingkat kesulitan permainan. Berdasarkan itu tentukan apakah di klaster itu duri akan di-*spawn*.



**Gambar 3.7** Contoh *Prefab* Ruang Dengan *Spike Tiles*



**Gambar 3.8** Hasil *Spawn Spike Prefab* Sebelumnya

#### 3.4.5. *Enemy Generator*

Musuh yang tersedia di dalam permainan ini terbagi menjadi 3 kategori: *normal*, *elite*, dan *boss*. Tingkat kesulitan melawan musuh meningkat dari tipe *normal* ke *elite* sampai *boss*, sehingga *reward* yang didapatkan dari mengalahkannya juga semakin besar.

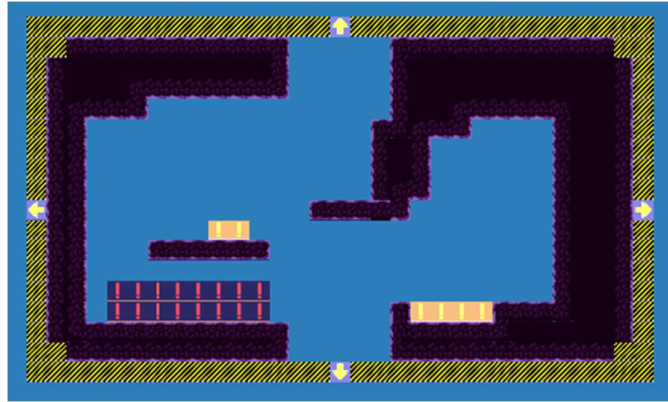
Setiap level hanya memiliki satu *boss enemy*, yang jika dikalahkan akan membuka pintu keluar dari level. *Boss enemy* yang akan di-*spawn* dipilih secara *random* dari daftar *boss* yang tersedia dan diposisikan di *bossTilePos* yang didapat dari proses *spawning level* sebelumnya. Setelah itu kondisi *boss* akan dipantau, ketika *boss* dikalahkan, level *exit* akan di-*spawn* di *exitTilePos*.



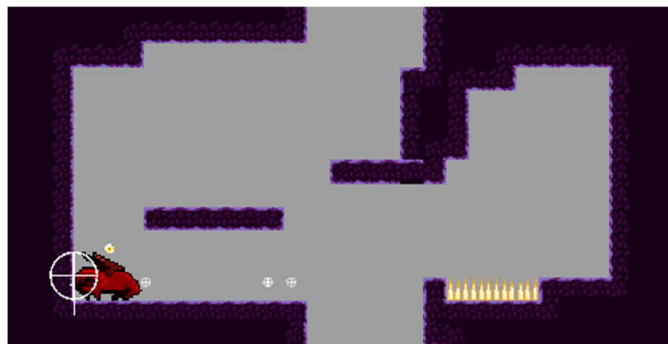
**Gambar 3.9** Contoh Musuh *Boss* (Indikator Tengkorak)

Adapun untuk *normal* dan *elite enemy* dapat muncul beberapa dalam level sehingga metode spawn mereka lebih kompleks. Alur *spawning* musuh untuk tipe *normal* dan *elite* dapat dijelaskan sebagai berikut:

- Mengelompokkan tile musuh yang berdekatan, membentuk klaster. Lokasi semua *tile* musuh didapatkan dari variabel *enemyTilePos* yang didapatkan saat *spawn* level.
- Untuk setiap kluster, hitung probabilitas pemunculan musuh (*spawnProbability*) berdasarkan kesulitan level.
- Musuh akan di-*spawn* di lokasi salah satu *tile* dalam klaster *tile*. Jenis musuh yang muncul, apakah *normal* atau *elite*, ditentukan berdasarkan hasil probabilitas yang sudah di kalkulasi.



**Gambar 3.10** Contoh *Prefab* Ruangn dengan *Enemy Tiles*



**Gambar 3.11** Hasil Level Dengan *Elite Enemy*

#### 3.4.6. *Item Generator*

*Item* yang dapat muncul dalam permainan ini dikategorikan berdasarkan harga dan kekuatan efeknya menjadi: *common*, *rare*, dan *legendary*. *Item* bisa didapatkan di *treasure* atau *shop event* dalam level.

##### 1) *Treasure Event*

Pada *treasure event* pemain bisa memilih 1 dari 3 *random item* dengan gratis. Posisi *treasure event* didapatkan dari *treasureSpawnPos* saat proses *spawning level*.

##### 2) *Shop Event*

Pada *shop event* pemain bisa membeli 3 *random item* dengan harga tertentu dibayar menggunakan *gold* yang didapatkan dari mengalahkan musuh. Posisi *shop event* didapatkan dari *shopSpawnPos* saat proses *spawning level*.

Pemilihan *Item* untuk *Shop/Treasure* dipengaruhi *multiplier* tiap *rarity item*. Berikut adalah tabel faktor pengali (*rarityMultiplier*) untuk setiap tingkat *rarity item*:



**Tabel 3.4** *Multiplier Rarity Item*

<i>Rarity</i>	<i>Multiplier</i>
<i>COMMON</i>	2.0
<i>RARE</i>	1.5
<i>LEGENDARY</i>	0.75

Faktor ini digunakan untuk mengatur probabilitas berdasarkan *rarity item*. *Item* dengan *rarity legendary* memiliki *multiplier* 0.75, sehingga memiliki kemungkinan lebih kecil untuk dipilih. Hasil kalkulasi probabilitas digunakan dalam pemilihan item harta dengan mempertimbangkan faktor-faktor berikut:

- **Probabilitas Tinggi:** Semakin tinggi probabilitas, semakin besar kemungkinan *item* dipilih.
- **Random Selection:** Proses pemilihan *item* dilakukan secara acak berdasarkan probabilitas yang dihitung.
- **Batasan Jumlah:** Hanya tiga *item* yang akan dipilih dari *pool item* yang tersedia.
- **Rarity Influence:** *Rarity item* berpengaruh pada faktor probabilitas melalui *multiplier*, memastikan *item-item* ber-*rarity* rendah lebih mungkin terpilih.

Dengan pendekatan ini, pemilihan item harta menjadi dinamis dan dipengaruhi oleh *rarity*, tingkat kesulitan permainan, dan keberuntungan.

### 3.4.7. *Player Spawn*

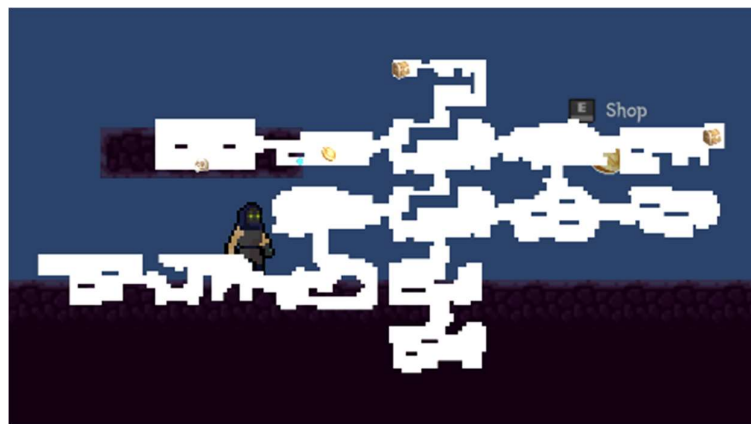
Di akhir, setelah semua proses pembuatan level selesai, karakter pemain diaktifkan dan dipindahkan ke *playerSpawnPos*. Setelah itu sesi permainan bisa dimulai.

## 3.5. *Desain User Interface*

Karena permainan ini didesain untuk menggunakan *PCG* pendekatan *hybrid*, setiap kali pemain memulai sesi permainan baru, konfigurasi ruangan dalam level akan selalu berubah. Untuk mempercepat alur permainan, pemain perlu mendapatkan bantuan dalam menavigasi level. Salah satu bantuan navigasi yang dapat diberikan adalah dalam bentuk *map/minimap UI* untuk memberikan sudut pandang luas terhadap struktur level bagi pemain dengan mudah.



**Gambar 3.12** *UI Minimap dan Gold Player*



**Gambar 3.13** *UI Full Map*

### 3.6. Rencana Pengujian

Pengujian akan dilakukan menggunakan kuisioner. Pertanyaan di dalam kuisioner akan dibagi ke dua kategori:

#### 3.6.1. Analisis Tingkat Kesulitan Level

Pengujian dilakukan untuk menguji rumus keseimbangan kesulitan level permainan dengan melihat dan menganalisis pengaruh parameter koefisien kesulitan tiap aspek level dan kesulitan level tersebut.

### 3.6.2. Pengujian Level Oleh *Playtester*

Pengujian dilakukan menggunakan kuisioner *online* dengan target orang-orang yang familiar dan berpengalaman dengan *video game* dan telah memainkan permainan yang dibuat peneliti. Pertanyaan di dalam kuisioner akan dibagi ke dua kategori. Pertama, dilakukan perbandingan hasil struktur level dengan satu tema tertentu yang dibuat dengan berbagai metode *procedural content generation*. Setelah itu, dilakukan penilaian mengenai berbagai aspek level yang dibuat menggunakan pendekatan hybrid *procedural content generation*. Seperti konsistensi level, *obstacle*, musuh, dan lain lain.

## BAB IV

### IMPLEMENTASI DAN PENGUJIAN SISTEM

#### 4.1. Video Game “Lost Labyrinths: Rogue’s Odyssey”

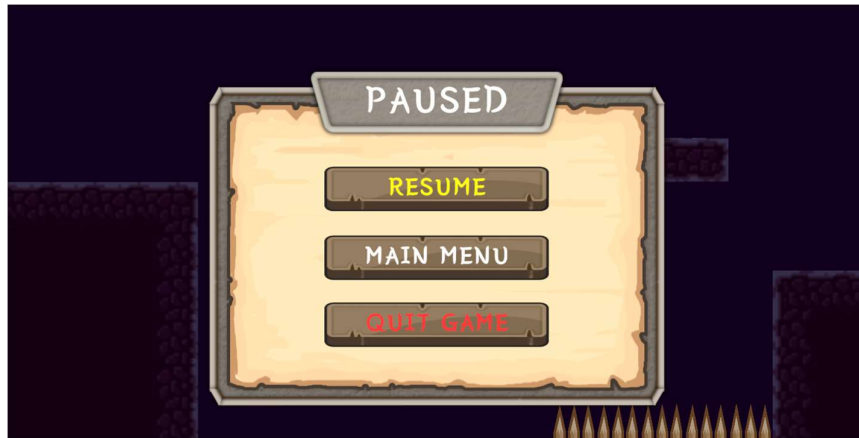
*Lost Labyrinth: Rogue’s Odyssey* adalah judul permainan yang dikembangkan oleh peneliti menggunakan implementasi *procedural content generation* pendekatan *hybrid* untuk pembuatan levelnya. Permainan ini memiliki *genre 2D action platformer*.



Gambar 4.1 Tampilan Main Menu Game

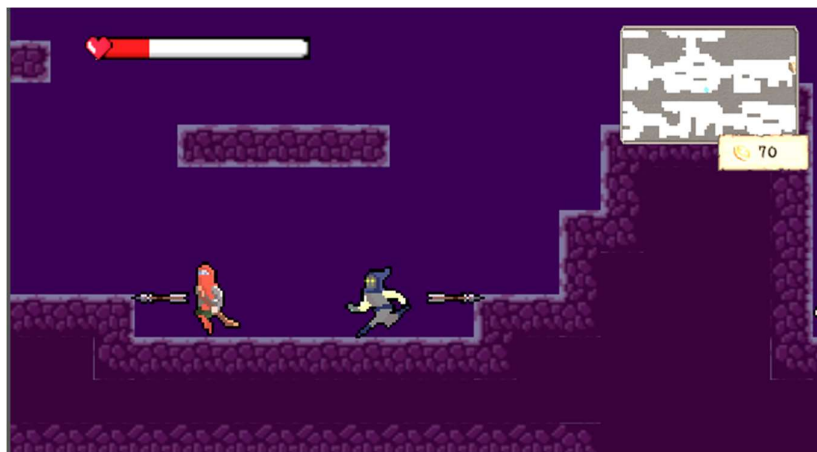


Gambar 4.2 Tampilan Settings dan Guide Game

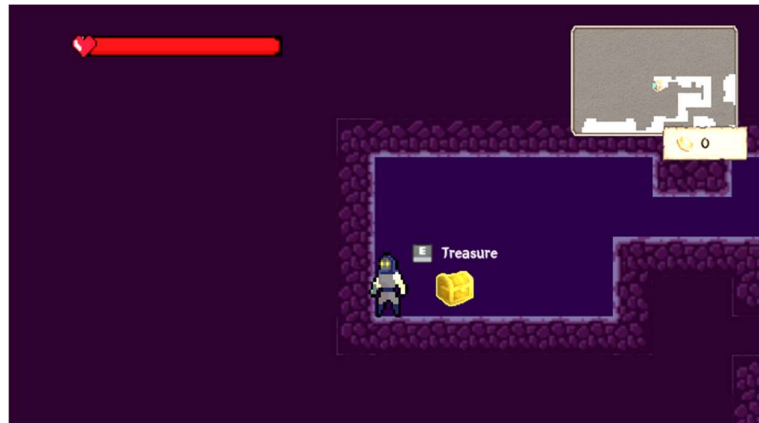


**Gambar 4.3** Tampilan *Pause Game*

Setiap kali permainan dimulai, level akan dibuat baru secara *randomized*, sehingga pemain tidak akan mudah bosan. Dalam level, pemain memiliki tujuan akhir mengalahkan *boss* dan keluar dari level. Di dalam level, pemain dapat menemukan berbagai musuh, rintangan, harta karun, dan *shop* yang semuanya di-*randomized*. Mengalahkan musuh dapat memberikan *gold* bagi pemain yang bisa digunakan dalam *shop*. Apabila nyawa pemain mencapai 0, maka sesi permainan selesai.



**Gambar 4.4** *Gameplay Action Game*



**Gambar 4.5** *Treasure Event Dalam Game*



**Gambar 4.6** *UI Treasure Game*



**Gambar 4.7** *Shop Event Dalam Game*



**Gambar 4.8** *UI Shop Game*

Untuk pengujian *procedural content generation* pendekatan *hybrid* dalam permainan ini, dibuat satu level dengan sebuah tema “Cave”. Penentuan tema dilakukan untuk nantinya dilakukan perbandingan dengan metode *PCG* lain. Untuk tingkat kesulitan (1-100) ditetapkan pada level *medium* (50).



**Gambar 4.9** Level Bertema "Cave"



**Gambar 4.10** *UI Map* untuk Membantu Navigasi Level

## 4.2. Pengujian Program

### 4.2.1. Analisis Tingkat Kesulitan Level

Dalam *Level Generator* secara keseluruhan, tingkat kesulitan level (*difficulty*) direpresentasikan ke dalam range angka dari 0 – 100. Tingkat kesulitan ini mempengaruhi aspek-aspek level:

#### 1) Jumlah Ruangan Level

Jumlah ruangan dalam level didefinisikan dengan rumus:

$$Srn = \text{Mathf.Clamp}\left(\frac{\text{difficulty}}{5}, \text{minRoom}, \text{maxRoom}\right) \quad (1)$$

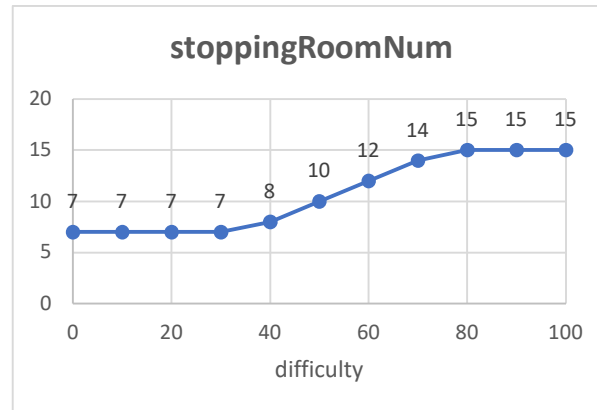
***stoppingRoomNum (Srn)***: Jumlah ruangan sebelum berhenti menambahkan ruangan koridor ke level.

***minRoom***: Jumlah minimal ruangan sebelum berhenti menambahkan ruangan koridor ke level.

***maxRoom***: Jumlah maksimal ruangan sebelum berhenti menambahkan ruangan koridor ke level.

Nilai *minRoom* dan *maxRoom* bisa dikonfigurasi oleh pengembang. Untuk pengujian kali ini *minRoom* akan dikonfigurasi menjadi 7 dan *maxRoom* menjadi 15. Nilai *stoppingRoomNum* bisa dipetakan dalam grafik di bawah ini.





**Gambar 4.11** Grafik Pengaruh *difficulty* Terhadap *stoppingRoomNum*

Dari hasil visualisasi ini keseimbangan besarnya level di dalam *game* dinilai **sudah seimbang**, dengan dibatasi tidak terlalu kecil atau besar, dan semakin tinggi kesulitan, semakin besar juga ukuran level.

## 2) Jumlah Duri / *Obstacle*

Duri dalam level muncul dalam grup, beranggotakan 4-6 duri. Untuk level dengan tingkat kesulitan normal (*difficulty* = 50) biasanya memiliki 30-40 grup posisi yang mungkin diisi oleh duri, Data didapatkan dari observasi dan analisis data menggunakan *build* permainan yang sudah dibuat. Adapun apakah grup akan diisi duri atau tidak ditentukan oleh rumus dibawah ini.

$$Sps = \text{Mathf.Clamp01} \left( \frac{\text{difficulty}}{100} * \frac{\text{tileCount}}{100} * \text{spikeC} \right) \quad (2)$$

***spawnProbabilitySpike (Sps)***: Kemungkinan duri muncul di satu grup

***tileCount***: Jumlah *tile* di satu grup

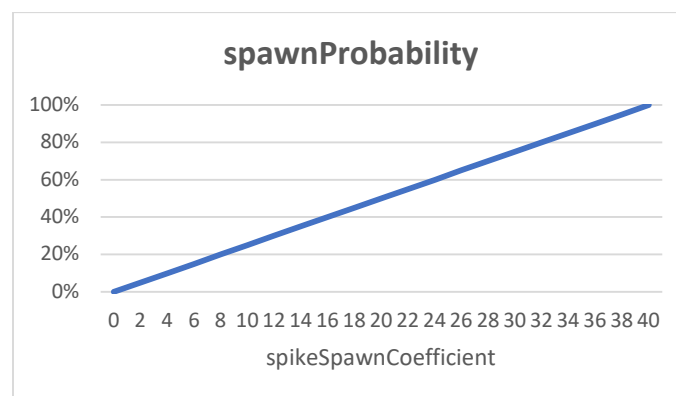
***difficulty***: Tingkat kesulitan level (0-100)

***spikeSpawnCoefficient (spikeC)***: Koefisien untuk membantu konfigurasi probabilitas munculnya duri di level (0 - 40). Bisa dikonfigurasi sesuai kebutuhan pengembang.

Untuk uji coba, digunakan data rata-rata dengan 35 grup duri, setiap grup berisi 5 *tile*. Berikut hasil coba untuk mendapatkan nilai *spikeSpawnCoefficient* paling seimbang.

**Tabel 4.1** Hasil Uji Coba Rumus *Spawn Spike*

<i>Spike Spawn Coefficient</i>	<i>Spawn Probability</i>	<i>Spike Group Muncul</i>
0	0%	0
2	5%	1.75
4	10%	3.5
6	15%	5.25
8	20%	7
10	25%	8.75
12	30%	10.5
14	35%	12.25
16	40%	14
<b>18</b>	<b>45%</b>	<b>15.75</b>
20	50%	17.5
22	55%	19.25
24	60%	21
26	65%	22.75
28	70%	24.5
30	75%	26.25
32	80%	28
34	85%	29.75
36	90%	31.5
38	95%	33.25
40	100%	35



**Gambar 4.12** Grafik Uji Coba Rumus *Spawn Spike*

Dari uji coba ini, didapatkan nilai *spikeSpawnCoefficient* dengan kesulitan normal paling seimbang adalah **18** dengan *spawn rate* duri sebesar **45%**.

### 3) Jumlah dan Tipe Musuh / *Enemy*

Musuh tipe *normal* dan *elite* muncul secara *randomized* di level, berbeda dengan musuh *boss* yang pasti hanya muncul satu dalam level. Untuk level dengan tingkat kesulitan normal (*difficulty* = 50) biasanya memiliki 12-20 grup posisi yang mungkin diisi oleh musuh, dan setiap grup berisi 8-12 *enemy tile*. Data didapatkan dari observasi dan analisis data menggunakan *build* permainan yang sudah dibuat. Adapun apakah grup akan diisi duri atau tidak ditentukan oleh rumus dibawah ini.

$$Spe = \text{Mathf.Clamp01} \left( \frac{\text{tileCount}}{100} * \frac{\text{difficulty}}{100} * \text{enemyC} \right) * 1.2$$

(3)

***spawnProbabilityEnemy (Spe)***: Kemungkinan musuh muncul di satu grup.

Apabila nilai lebih dari 100% maka musuh *elite* yang muncul

***tileCount***: Jumlah *tile* di satu grup

***difficulty***: Tingkat kesulitan level (0-100)

***enemySpawnCoefficient (enemyC)***: Koefisien untuk membantu konfigurasi probabilitas munculnya musuh di level (0 - 25). Bisa dikonfigurasi sesuai kebutuhan pengembang.

Hasil kalkulasi probabilitas akan menentukan musuh yang muncul dalam satu grup *enemy tile*, sesuai dengan tabel berikut.

**Tabel 4.2** Outcome Probabilitas *Spawn* Musuh

Probabilitas	Hasil
0% - 70%	Tidak Ada Musuh
70% - 100%	<i>Spawn</i> Musuh Normal
> 100%	<i>Spawn</i> Musuh <i>Elite</i>

Untuk uji coba, digunakan data rata-rata dengan setiap grup *enemy tile* berisi bervariasi dari 8 sampai 12 *tile*. Berikut hasil coba untuk mendapatkan nilai *spikeSpawnCoefficient* paling seimbang.

**Tabel 4.3** Hasil Uji Coba Rumus *Spawn Enemy*

<i>Enemy Spawn Coefficient</i>	<i>Spawn Probability</i>				
	<i>8 Tile</i>	<i>9 Tile</i>	<i>10 Tile</i>	<i>11 Tile</i>	<i>12 Tile</i>
0	0%	0%	0%	0%	0%
2	10%	11%	12%	13%	14%
4	19%	22%	24%	26%	29%
6	29%	32%	36%	40%	43%
8	38%	43%	48%	53%	58%
10	48%	54%	60%	66%	72%
12	58%	65%	72%	79%	86%
14	67%	76%	84%	92%	101%
<b>16</b>	<b>77%</b>	<b>86%</b>	<b>96%</b>	<b>106%</b>	<b>115%</b>
18	86%	97%	108%	119%	120%
20	96%	108%	120%	120%	120%
22	106%	119%	120%	120%	120%
24	115%	120%	120%	120%	120%
25	120%	120%	120%	120%	120%

Dari uji coba ini, didapatkan nilai *enemySpawnCoefficient* dengan kesulitan normal paling seimbang adalah **16** dengan *spawn rate* dari sebesar **77-115%** untuk jumlah *tile* di grup dari **8-12 tiles**.

#### 4) Jumlah Ruangan *Treasure*

*Item* bisa didapatkan dari ruangan *treasure* atau *shop*. Setiap level hanya memiliki satu ruangan *shop*, akan tetapi bisa memiliki beberapa ruangan *treasure*. Jumlah ruangan *treasure* dalam level bisa dikalkulasi dengan rumus di bawah ini.

$$Tts = \text{Mathf.Clamp}\left(\frac{Srn}{treasureC}, minRoom, maxRoom\right)$$

(4)

*treasureToSpawn (Tts)*: Jumlah ruangan *treasure* dalam level

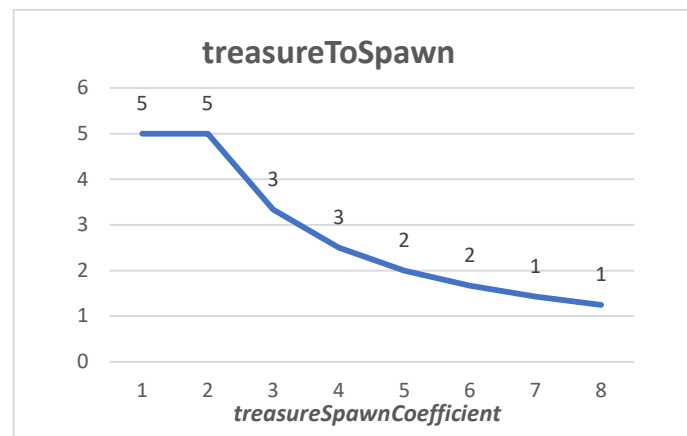
***minRoom***: Jumlah minimal ruangan *treasure* dalam level. Bisa dikonfigurasi pengembang

***maxRoom***: Jumlah maksimal ruangan *treasure* dalam level. Bisa dikonfigurasi pengembang

***stoppingRoomNum (Srn)***: Jumlah ruangan sebelum berhenti menambahkan ruangan koridor ke level.

***treasureSpawnCoefficient (treasureC)***: Koefisien untuk membantu konfigurasi probabilitas munculnya *treasure room* di level (0 - 25). Bisa dikonfigurasi sesuai kebutuhan pengembang.

Uji coba dilakukan menggunakan *stoppingRoomNum* dengan nilai 10 sesuai dengan hasil uji coba jumlah ruangan dalam level medium (*difficulty* = 50) sebelumnya.



**Gambar 4.13** Pengaruh *treasureSpawnCoefficient* Pada Jumlah *Treasure*

Dari uji coba ini, didapatkan nilai *treasureSpawnCoefficient* dengan kesulitan normal paling seimbang adalah 4 dengan jumlah ruangan *treasure* sebesar 3.

### 5) *Item dalam Treasure dan Shop*

*Item* dalam event *treasure* atau *shop* muncul dalam kumpulan 3 *item*. *Item* memiliki 3 *rarity*: *common*, *rare*, dan *legendary*. Probabilitas muncul setiap *item rarity* dipengaruhi *spawn multiplier* tiap *item*:

**Tabel 4.4** *Multiplier Rarity Item*

<b><i>Rarity</i></b>	<b><i>Multiplier</i></b>
<i>COMMON</i>	2.0
<i>RARE</i>	1.5
<i>LEGENDARY</i>	0.75

Probabilitas muncul setiap *item* dikalkulasi dengan rumus:

$$Final\ Probability = Bp \times Rarity\ Multiplier \times Df$$

(5)

***Final Probability***: Nilai probabilitas *item* dengan *rarity* tertentu muncul dalam *shop/treasure*.

***Base Probability (Bp)***: Nilai dasar probabilitas, diatur pada 0.5.

***Rarity Multiplier***: Faktor pengali berdasarkan *rarity item*, diambil dari tabel di atas.

***Difficulty Factor (Df)***: Faktor kesulitan yang dinormalisasi (nilai dalam *range* 0 - 1).

Berdasarkan rumus tersebut, kemungkinan muncul dalam *shop/treasure* semua *item* berdasarkan *rarity* bisa dilihat dalam tabel di bawah ini.

**Tabel 4.5** Hasil Uji Coba Rumus *Spawn Item*

<b><i>Difficulty</i></b>	<b><i>Item Probability di Shop / Treasure</i></b>		
	<b><i>Common</i></b>	<b><i>Rare</i></b>	<b><i>Legendary</i></b>
10	10.00%	7.50%	3.75%
20	20.00%	15.00%	7.50%
30	30.00%	22.50%	11.25%
40	40.00%	30.00%	15.00%
<b>50</b>	<b>50.00%</b>	<b>37.50%</b>	<b>18.75%</b>
60	60.00%	45.00%	22.50%
70	70.00%	52.50%	26.25%
80	80.00%	60.00%	30.00%
90	90.00%	67.50%	33.75%
100	100.00%	75.00%	37.50%

Dari hasil visualisasi ini keseimbangan *rarity item* di dalam *game* dinilai **sudah seimbang**,

#### 4.2.2. Hasil Pengujian Oleh Sampel

Peneliti memberikan akses kepada 6 orang untuk memainkan permainan yang sudah dibuat peneliti. Semua anggota sampel familiar dan juga berpengalaman di dunia *video game*. Setelah bermain selama 15 menit, semua sampel diberikan kuesioner mengenai level yang dibuat dengan *procedural content generation* serta pengalaman bermain sampel.

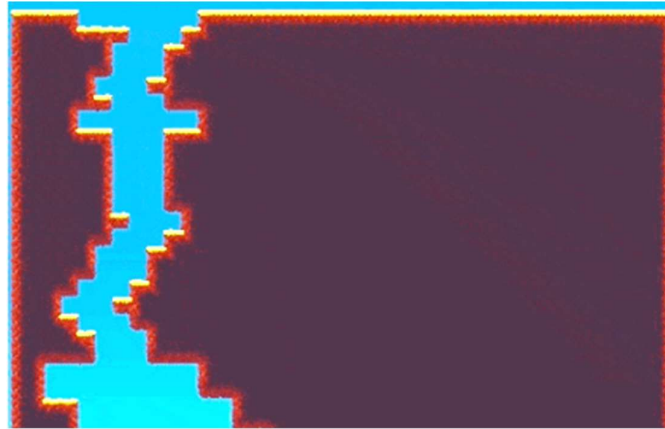
Bagian pertama dari kuisisioner membahas mengenai level yang dibuat menggunakan berbagai metode *procedural content generation*. Pertanyaan membandingkan berbagai hasil level yang dibuat dengan berbagai metode dengan tema yang sama, yaitu “cave”. Adapun metode yang dibandingkan yaitu: a) algoritma *perlin noise*, b) algoritma *random walk*, c) algoritma *directional tunnel*, d) algoritma *cellular automata*, dan e) metode *hybrid*.



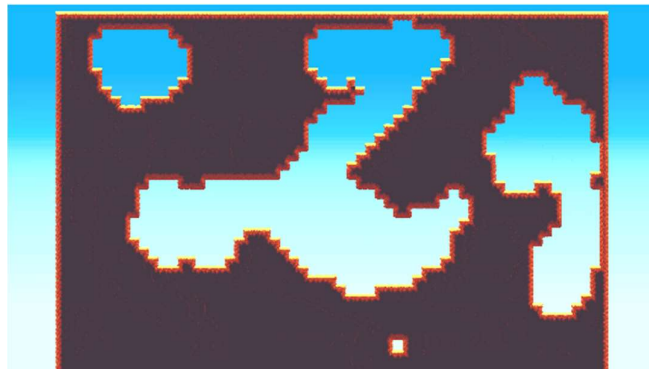
**Gambar 4.14** Level “Cave” PCG Perlin Noise (a),  
Sumber: Ethan Bruins – *Unity Blog*



**Gambar 4.15** Level “Cave” PCG Random Walk (b),  
Sumber: Ethan Bruins – *Unity Blog*



**Gambar 4.16** Level “Cave” *PCG Directional Tunnel* (c),  
Sumber: Ethan Bruins – *Unity Blog*



**Gambar 4.17** Level “Cave” *PCG Cellular Automata* (d),  
Sumber: Ethan Bruins – *Unity Blog*



**Gambar 4.18** Level “Cave” *PCG Pendekatan Hybrid* (e)



**Tabel 4.6** Hasil Kuisioner Perbandingan Metode *PCG*

Metode <i>PCG</i>	Hasil					Total Poin	Rata Rata	Kesimpulan
	Sangat Baik (5)	Baik (4)	Cukup (3)	Kurang (2)	Sangat Kurang (1)			
<i>Perlin Noise</i>	0	2	1	1	2	15	2.5	Cukup
<i>Random Walk</i>	0	0	0	3	3	9	1.5	Kurang
<i>Directional Tunnel</i>	2	2	2	0	0	24	4	Baik
<i>Cellular Automata</i>	0	1	1	3	1	14	2.333	Kurang
Metode <i>Hybrid</i>	6	0	0	0	0	30	5	Sangat Baik

Dari hasil kuisioner ini, bisa dilihat metode *PCG* dengan hasil paling baik untuk membuat level permainan, terutama dengan tema tertentu, adalah dengan **metode *hybrid*** dengan hasil **sangat baik (5)**. Hal ini membuktikan keefektifan pembuatan level menggunakan pendekatan *hybrid* dibandingkan dengan metode *PCG* lainnya.

Di set pertanyaan kuisioner berikutnya, sampel akan menilai aspek-aspek dalam permainan yang dibuat peneliti yang berhubungan dengan implementasi *procedural content generation* metode *hybrid* dalam pembuatan level.

**Tabel 4.7** Hasil Kuisioner Aspek Game

Aspek <i>Game</i>	Hasil					Total Poin	Rata Rata	Kesimpulan
	Sangat Baik (5)	Baik (4)	Cukup (3)	Kurang (2)	Sangat Kurang (1)			
Keseuaian estetika level dengan tema level	4	2	0	0	0	28	4.667	Sangat Baik
Konsistensi visual semua ruangan dengan tema level	5	1	0	0	0	29	4.833	Sangat Baik

Distribusi <i>obstacle duri</i>	4	2	0	0	0	28	4.667	Sangat Baik
Keseimbangan Jumlah dan Jenis Musuh	3	1	2	0	0	25	4.167	Baik
Jumlah dan <i>item treasure</i>	5	1	0	0	0	29	4.833	Sangat Baik
Jumlah dan <i>item shop</i>	5	1	0	0	0	29	4.833	Sangat Baik
Kemudahan navigasi level	5	1	0	0	0	29	4.833	Sangat Baik

Dari hasil kuisioner ini, bisa dilihat untuk semua aspek level permainan yang telah dibuat memiliki hasil **sangat baik**, kecuali aspek keseimbangan jumlah dan jenis musuh dengan hasil **baik**. Hal ini membuktikan **kesuksesan** pembuatan permainan menggunakan *PCG* pendekatan *hybrid*.

## BAB V

### KESIMPULAN DAN SARAN

#### 5.1. Kesimpulan

Berdasarkan penelitian pengimplementasian *procedural content generation* pendekatan *hybrid* dalam pembuatan level dalam *game*, dapat disimpulkan:

- 1) Implementasi *procedural content generation* pendekatan *hybrid* bisa dan sangat baik dalam mengatasi kelemahan *procedural content generation* biasa, seperti penurunan kualitas desain level serta kurangnya perasaan konsistensi dan kesesuaian estetika visual dalam level dengan tema tertentu pada *game*.
- 2) Berdasarkan hasil pengujian *game tester*, implementasi *procedural content generation* pendekatan *hybrid* dalam pembuatan level *game* dengan tema tertentu, lebih baik dalam menghasilkan level yang memiliki kesesuaian estetika visual dan konsistensi sangat baik daripada metode *PCG* lainnya, seperti: *perlin noise*, *random walk*, *directional tunnel*, dan *cellular automata*.
- 3) Berdasarkan pengalaman *playtester* dalam memainkan *game* yang dibuat peneliti, implementasi *procedural content generation* pendekatan *hybrid* dalam pembuatan *game*, juga bisa menghasilkan aspek-aspek *procedural* lainnya dengan sangat baik, seperti: penyebaran *obstacle* dan musuh dalam level serta mengatur munculnya ruangan atau *item* tertentu dalam permainan.

#### 5.2. Saran

Adapun beberapa saran yang bisa diberikan setelah penelitian ini, yaitu:

- 1) Generasi berikutnya untuk bisa melanjutkan penelitian ini bisa meningkatkan hasil yang masih bisa disempurnakan, seperti aspek keseimbangan jumlah dan jenis musuh dalam level.
- 2) Permainan yang dibuat masih bisa disempurnakan lagi, mulai dari aspek dekorasi level, *UI*, *audio*, *story*, hingga tambahan level dengan tema lain sehingga permainan bisa menjadi lebih baik lagi.

## DAFTAR PUSTAKA

- Baghdadi, W., Shams Eddin, F., Al-Omari, R., Alhalawani, Z., Shaker, M., & Shaker, N. (2015). A Procedural Method for Automatic Generation of Spelunky Levels. *European Conference on the Applications of Evolutionary Computation*, 305–317.
- Bardent. (2022). *The Bardent Asset Pack!*. <https://bardent.itch.io/the-bardent-asset-pack>
- Benard, S. (2017, Maret 29). *Building the Level Design of a procedurally generated Metroidvania: a hybrid approach*. <https://www.gamedeveloper.com/design/building-the-level-design-of-a-procedurally-generated-metroidvania-a-hybrid-approach->
- Bruins, E. (2018). *Procedural patterns to use with Tilemaps, part 2*. <https://blog.unity.com/engine-platform/procedural-patterns-to-use-with-tilemaps-part-2>
- Bycer, J. (2023). Game Design Deep Dive. Dalam *Game Design Deep Dive*. CRC Press. <https://doi.org/10.1201/9781003335214>
- Cook, M., Gow, J., Smith, G., & Colton, S. (2021). Danesh: Interactive Tools For Understanding Procedural Content Generators. *IEEE Transactions on Games*, 14(3), 329–338. <https://unity3d.com/public-relations>
- Gellel, A., & Sweetser, P. (2020, September 15). A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3402942.3402945>
- Ripamonti, L. A., Mannalà, M., Gadia, D., & Maggiorini, D. (2017). Procedural content generation for platformers: designing and testing FUN PLEdGE. *Multimedia Tools and Applications*, 76(4), 5001–5050. <https://doi.org/10.1007/s11042-016-3636-3>
- Risi, S., & Togelius, J. (2019). *Increasing Generality in Machine Learning through Procedural Content Generation*. <http://arxiv.org/abs/1911.13071>
- Shaker, N., Togelius, J., & Nelson, M. (2015). Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Dalam *Retrogame Archeology*.
- Shan, Y. (2021). *A procedural character generation system* [Master's Thesis, Aalto University]. [www.aalto.fi](http://www.aalto.fi)

- Silva, R. C. E., Fachada, N., De Andrade, D., & Codices, N. (2022). Procedural Generation of 3D Maps With Snappable Meshes. *IEEE Access*, 10, 43093–43111. <https://doi.org/10.1109/ACCESS.2022.3168832>
- Watkins, R. (2016). *Procedural content generation for Unity game development : harness the power of procedural content generation to design unique games with Unity* (V. Arora, M. Pereira, M. Mathew, & P. Bisht, Ed.; 1st ed.). Packt Publishing.s

## LAMPIRAN

### KUISIONER PERBANDINGAN METODE PROCEDURAL CONTENT GENERATION DAN PENGALAMAN BERMAIN GAME “LOST LABYRINTHS: ROGUE’S ODYSSEY”

**Sangat Kurang = 1, Kurang = 2, Cukup = 3, Baik = 4, Sangat Baik = 5**

No.	Pertanyaan	1	2	3	4	5
1	Bagaimana penilaian Anda terhadap struktur map yang dihasilkan menggunakan algoritma Perlin Noise untuk tema "cave"?					
2	Bagaimana penilaian Anda terhadap struktur map yang dihasilkan menggunakan algoritma Random Walk untuk tema "cave"?					
3	Bagaimana penilaian Anda terhadap struktur map yang dihasilkan menggunakan algoritma Directional Tunnel untuk tema "cave"?					
4	Bagaimana penilaian Anda terhadap struktur map yang dihasilkan menggunakan algoritma Cellular Automata untuk tema "cave"?					
5	Bagaimana penilaian Anda terhadap struktur map yang dihasilkan menggunakan Procedural Content Generation metode hybrid untuk tema "cave"?					
6	Menurut anda bagaimana kesesuaian estetika level dengan tema "Cave" dalam game ini?					
7	Menurut anda bagaimana perasaan konsistensi visual semua ruangan dengan tema "Cave" dalam game ini?					
8	Menurut anda bagaimana pengaruh distribusi duri terhadap keseimbangan kesulitan dalam game ini?					
9	Menurut anda bagaimana keseimbangan jumlah dan jenis (normal dan elite) musuh terhadap keseimbangan kesulitan dalam game ini?					
10	Menurut anda bagaimana keseimbangan jumlah ruangan harta karun dan pilihan item di dalamnya terhadap keseimbangan kesulitan dalam game ini?					
11	Menurut anda bagaimana keseimbangan jumlah ruangan shop dan pilihan item di dalamnya terhadap keseimbangan kesulitan dalam game ini?					
12	Menurut anda bagaimana tingkat kemudahan navigasi level dalam game ini?					

### TABULASI JAWABAN RESPONDEN

<b>Responden</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
<b>A</b>	3	1	3	2	5	5	5	4	3	5	5	5
<b>B</b>	4	1	4	2	5	4	5	5	3	5	5	5
<b>C</b>	4	2	3	4	5	5	5	5	5	5	5	5
<b>D</b>	1	1	4	1	5	5	5	5	5	5	5	5
<b>E</b>	2	2	5	3	5	4	4	4	4	4	4	4
<b>F</b>	1	2	5	2	5	5	5	5	5	5	5	5