Jurnal Teknik Informatika (JUTIF)
Vol. 5, No. 3, June 2024, pp. 873-881

DOI: https://doi.org/10.52436/1.jutif.2024.5.3.1465
p-ISSN: 2723-3863
e-ISSN: 2723-3871

APPLICATION OF PROCEDURAL CONTENT GENERATION SYSTEM IN FORMING DUNGEON LEVEL IN DUNGEON DIVER GAME

Eka Wahyu Hidayat[*1], Euis Nur Fitriani Dewi[2], Insan Saleh Ramadhan[3]

[1,2,3]Department of Informatics, Faculty of Engineering, Siliwangi University, Indonesia
Email: [1]ekawahyu@unsil.ac.id, [2]euis.nurfitriani@unsil.ac.id, [3]187006032@student.unsil.ac.id

(Article received: October 11, 2023; Revision: November 20, 2023; published: June 05, 2024)

Abstract

Developers face numerous challenges in game development, one of which is the lack of games replayability due to the limited variety of levels created. The absence of level variety can lead to player boredom. The Procedural Content Generation (PCG) method provides an effective solution to address this challenge. PCG is applied with a focus on the Cellular Automata method by implementing the Von Neumann Neighborhood rule. The objective of this paper is to apply the Procedural Content Generation System method to create levels in game development. The game development process utilizes Luther's MDLC method. Testing is conducted using tiles of 32x32 units and 64x64 units, with three different test parameters: a fill percentage of 25%, 45%, and 65%. Each fill percentage is tested with three different smooth amount parameters of 2, 4, and 6, with a randomly selected seed. Performance testing results indicate that creating dungeon levels with 32x32 and 64x64 tiles yields short and relatively similar times, around 0.08 to 0.3 seconds. Functional testing reveals that a 25% fill percentage results in nearly empty rooms with no footholds, a 45% fill percentage produces levels with space and footholds, while a 65% fill percentage generates small unconnected rooms. Based on these percentages, a 45% fill percentage is considered the most appropriate for creating dungeon levels because it provides suitable space and footholds for players. Implementing PCG in game level creation not only saves time compared to manual level creation but also offers more efficient variations in dungeon shapes and difficulty levels.

Keywords: *Cellular Automata, Games, Levels, Procedural Content Generation.*

PENERAPAN PROCEDURAL CONTENT GENERATION SYSTEM DALAM MEMBENTUK DUNGEON LEVEL PADA *GAME* DUNGEON DIVER

Abstrak

Terdapat banyak tantangan yang dihadapi oleh para developer dalam pengembangan game, salah satunya adalah kurangnya daya replayabilty game karena kurangnya variasi level yang dibuat. Kurangnya variasi level dapat membuat pemain bosan memainkan game. Metode Procedural Content Generation (PCG) menjadi solusi efektif untuk mengatasi tantangan ini. PCG diterapkan dengan fokus pada metode Cellular Automata dengan menerapkan aturan Von Neumann Neighborhood. Tujuan makalah ini adalah mengaplikasikan metode Procedural Content Generation System untuk menciptakan level dalam pengembangan game. Proses pengembangan game ini menggunakan metode MDLC versi Luther. Pengujian dilakukan dengan menggunakan tile berukuran 32x32 unit dan 64x64 unit, dengan 3 parameter uji yang berbeda, yaitu fill percentage sebesar 25%, 45%, dan 65%. Setiap fill percentage diuji dengan 3 parameter smooth amount sebesar 2, 4, dan 6, dengan seed yang dipilih secara acak. Hasil pengujian performa menunjukkan bahwa pembuatan level dungeon dengan tile 32x32 dan 64x64 menghasilkan waktu yang singkat dan relatif serupa, yakni sekitar 0.08 detik hingga 0.3 detik. Pengujian fungsional menunjukkan fill percentage 25% menghasilkan ruangan yang hampir kosong tanpa pijakan, fill percentage 45% menghasilkan level dengan ruang dan pijakan, sementara fill percentage 65% menghasilkan level dengan ruangan kecil yang tidak terhubung. Dari persentase tersebut, fill percentage sebesar 45% dianggap paling tepat digunakan dalam meghasilkan level dungeon, karena fill percentage tersebut memberikan ruang dan pijakan yang pas untuk pemain. Penerapan PCG dalam pembuatan level game bukan hanya dapat menghemat waktu dalam pembuatan level secara manual, tetapi juga dapat memberikan variasi bentuk dungeon dan tingkat kesulitan secara lebih efisien.

Kata kunci: *Cellular Automata, Game, Level, Procedural Content Generation.*

1. PENDAHULUAN

Selaras dengan perkembangan teknologi, industri game sebagai bagian dari industri kreatif menghadapi tantangan pada konsep pengembangannya. Salah satunya yaitu daya *replayability* sebuah *game* karena kurangnya variasi *level* pada suatu *game* [1]. Pemain biasanya cepat merasa bosan saat harus menjelajahi *level* yang sama secara terus menerus, sehingga desainer *game* harus mendesain *level* semenarik mungkin, seperti menambahkan variasi tata letak dan tantangan *level*. *Level* yang tidak bervariasi dapat membuat pemain cepat merasa bosan sehingga menjadi tugas desainer *game* untuk menciptakan variasi tata letak dan tantangan *level* [2]. Sebagai desainer *game*, menciptakan peta atau *level* bervariasi dalam jumlah banyak bisa membuat jenuh dan memakan waktu yang cukup lama. Otomatisasi pada pembuatan *level* memungkinkan pembuatan variasi *level* yang hampir tak terbatas.

Procedural Content Generation (PCG) dapat digunakan untuk menyelesaikan masalah tersebut tersebut. PCG sendiri merujuk pada istilah pembuatan konten *game* yang dilakukan secara algoritmis, baik itu dalam pembuatan peta, *level*, *quest*, karakter, dan lain-lain [3]. PCG dapat membantu dalam eksplorasi dan mempercepat proses pembuatan konten [4]. menurut [5] PCG dapat digunakan untuk membuat otomatisasi level game.

Pada penelitian seperti [6] menjelaskan bahwa PCG dalam *game* memliki sejarah panjang, dengan berbagai metode pendekatan yang bervariatif. Pendekatan yang pernah dilakukan diantaranya menggunakan metode *Deep Learning*. Melakukan kombinasi atau metode *Hybrid* [7]. Serta pendekatan tunggal lain seperti *Binary Tree* [8], *Context-Free Grammar* [9] dan *Genetic Algorithm* [10]. Pembuatan *level* secara prosedural juga bisa dilakukan dengan menggabungkan algoritma PCG dengan pembuatan *level* secara manual oleh desainer [11].

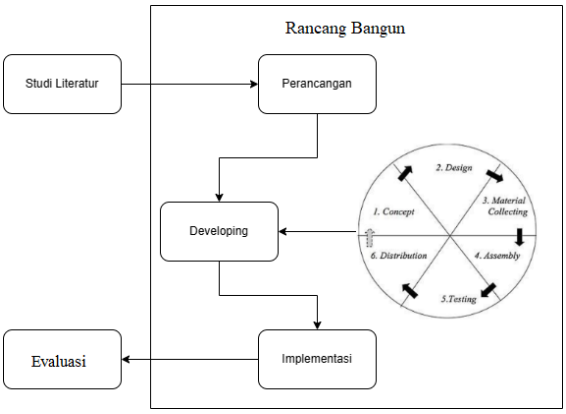
Dari metode PCG yang ada, metode *Cellular Automata* dirasa tepat untuk pembuatan *level dungeon*, karena teknik ini dirasa cepat, sederhana, memiliki biaya resource yang rendah, dan dapat menghasilkan lingkungan alam yang natural, layaknya sebuah gua bawah tanah [12]. PCG juga bisa diterapkan pada beberapa aspek *game* yang lain seperti membangkitkan sebuah skenario berbeda [13], alur *game* seperti pada *game visual novel* [14], *sub-system* tujuan atau *objective* [15], dan lain sebagainya.

Penelitian paling mendekati dengan penelitian ini yaitu penelitian yang dilakukan oleh [16] dimana penelitian tersebut memiliki masalah yang sama berupa pembuatan variasi *level*, efisiensi proses pembuatannya, serta *game* yang dibuat memiliki kesamaan berbasis *level* 2D. Perbedaan dengan penelitian tersebut yaitu tipe algoritma yang digunakan serta desain *level* yang ingin dicapai.

Berdasarkan pemaparan tersebut, dibuat sebuah *game platformer sidescrollling 2D* bergaya *dungeon/cavern level*, berjudul "Dungeon Diver", yang bertujuan menerapkan teknik PCG dalam membuat *level* pada rancang bangun *game*, untuk memudahkan para desainer membuat *level* yang dihasilkan secara otomatis menggunakan algoritma.

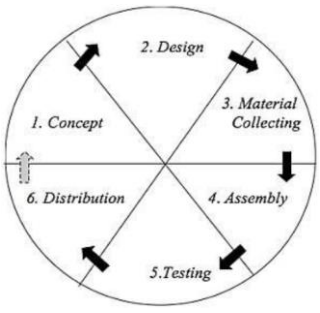
2. METODE

Tahapan penelitian yang dilakukan terdiri dari lima tahapan, seperti ditunjukkan pada Gambar 1 di bawah ini.



Gambar 1. Metode Penelitian

Secara inti, tahapan penelitian yang dilakukan yaitu studi literatur, rancang bangun, lalu kesimpulan. Tahap pertama adalah Studi literatur. Pada tahap ini ditelusuri dan diperdalam pemahaman terkait penerapan PCG dalam pengembangan video *game*, saat pembuatan laporan. Tahap rancang bangun sendiri terdiri dari 3 tahapan, yaitu perancangan, *developing*, serta implementasi. Pada tahap perancangan, dibuat perencanaan bagaimana *game* dikembangkan, termasuk material-material yang dibutuhkan dalam pengembangan *game* yang dibuat. Pada tahap ini, dilakukan analisis kebutuhan yang berkaitan dengan komponen esensial yang diperlukan dalam pengembangan *game*. Tahap berikutnya yaitu *Developing*. Pada tahap *developing*, digunakan metode pengembangan *Multimedia Development Life Cycle* (MDLC) oleh Luther, yang terdiri dari enam tahapan, yaitu *Concept*, *Design*, *Material Collecting*, *Assembly*, *Testing*, dan *Distribution* [17]. Diagram tahapan ini bisa dilihat pada Gambar 2 di bawah ini:



Gambar 2. Diagram MDLC [17].

Pada tahap implementasi, dilakukan proses penerapan PCG pada pembuatan *level* dungeon pada permainan yang dibangun. Tahap terakhir yaitu tahap evaluasi. Pada tahap evaluasi, dilakukan analisis terhadap hasil pengujian dari implementasi PCG dalam pembuatan *level* pada pengembangan *game* ini. Hal yang dievaluasi meliputi perbedaan waktu pembangkitan level tiap parameter yang diuji dan analisa perbedaan tata letak *level*.

3. HASIL DAN PEMBAHASAN

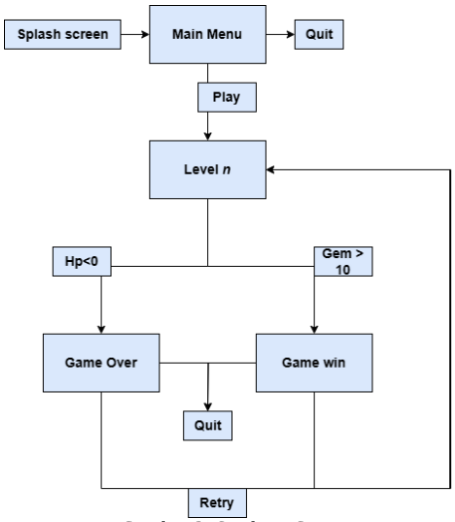
3.1. Studi Literatur

Pada tahap ini ditelusuri dan diperdalam pemahaman terkait penerapan PCG dalam pengembangan *game* saat pembuatan laporan. Dari hasil studi, didapatkan informasi seperti pengertian PCG, implementasi PCG pada *game*, metode PCG yang ada dan digunakan pada penelitian terdahulu, dan perbedaan hasil implementasi PCG pada *game* yang dibuat.

3.2. Perancangan

Game ini didesain dengan genre *sidescroller platformer*, di mana pemain mengendalikan karakter untuk berpindah-pindah dari satu platform ke platform lain dalam lingkungan 2D. Konsep ini dipilih untuk memberikan tantangan kepada pemain dalam mengatasi rintangan dan musuh di sepanjang perjalanan**.**

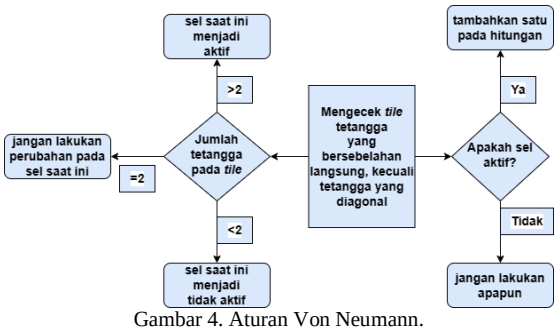
Struktur *game* yang dikembangkan dapat dilihat pada Gambar 3 di bawah ini:



Gambar 3. Struktur *Game*

Setelah melewati fase *Splash screen*, pemain diberi tampilan pada *main menu.* Disini pemain bisa menutup *game* dengan tombol *Quit*, atau memulai permainan dengan tombol *Play*. *Level* yang dibuka yaitu *Level*-n, yang berarti *level* dipilih akan muncul secara acak. Pemain akan dihadapkan dengan layar *Game Over* jika karakter pemain mati, dan layar *Game Win* jika pemain berhasil mengumpulkan gem yang cukup. Gem ini merupakan sebuah item koleksi yang diperlukan pemain untuk memenangkan suatu *level*.

Pembuatan *level game* ini dilakukan secara prosedural menggunakan metode *Cellular Automata*, dengan aturan *Von Neumann Neighborhood*. Implementasi metode tersebut menghasilkan tata letak *level* yang berbentuk kotak. Aturannya sendiri ditunjukkan pada Gambar 4 berikut:



Gambar 4. Aturan Von Neumann.

Implementasi PCG pada pembuatan *level* ini bergantung pada *smooth amount* yang berfungsi untuk memperhalus hasil *layout*, *Fill percentage* yang befungsi mengatur berapa persen sel aktif terisi dari total luas *level* yang dibuat, serta *seed* yang merujuk pada nilai awal atau input yang digunakan dalam algoritma yang menghasilkan data acak.

3.3. Development

Pada tahap *development*, diterapkan metode pengembangan MDLC oleh Luther. Tahapan-tahapannya adalah sebagai berikut:

A. Concept

Pada tahap *Concept*, dijelaskan konsep game yang dibangun. Game ini berfokus pada pergerakan karakter dari satu platform ke platform yang lain, mengumpulkan gem untuk menang, serta menghindari musuh yang bisa menyerang pemain dengan mendekati pemain secara langsung, maupun menembak pemain menggunakan proyektil.

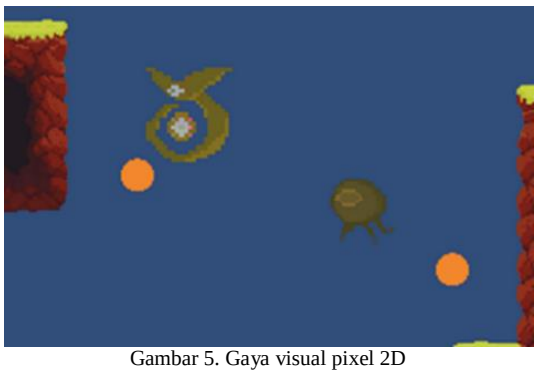
Level yang ada pada *game* ini, dibuat secara otomatis menggunakan algoritma PCG *Cellular automata* dengan aturan *Von Neuman Neighborhood*. *Game* ini dibangun pada perangkat lunak Unity.

Deskripsi lengkap *game* yang dikembangkan tertera pada Tabel 1 di bawah berikut:

Tabel 1. Deskripsi *game*.

Keterangan	Deskripsi
Judul	Dungeon Diver
Genre	*Platformer*
Grafik	2D *Pixelated*
Pergerakan karakter	Gerakan menyamping, ke atas, maupun ke bawah.
Kamera	Penglihatan nampak dari samping
Interaktif	Pergerakan *platformer* umum, menginjak musuh, membuka menu.
Suara	Musik latar belakang, efek suara lainnya.
Software pendukung	Unity, Aseprite, vs code
Target Audiens	Umum (Semua kalangan)
Platform	*PC Windows*
Monetisasi	Gratis

Grafis *game* bergaya pixel 2D dipilih untuk memberikan nuansa klasik dan mengingatkan pada *game-game platformer* retro. Perspektif kamera yang digunakan adalah kamera menyamping yang umum pada *game platformer* 2D. Gaya visual *game* yang ingin dicapai bisa dilihat pada Gambar 4 di bawah:



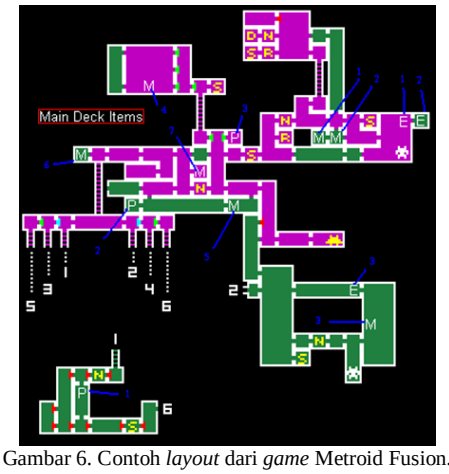
Gambar 5. Gaya visual pixel 2D

B. Design

Desain permainan ini yaitu *game* dengan genre *platformer*, bergaya Pixel 2D, dimana pemain mengendalikan sebuah karakter dimana karakter tersebut ditempatkan pada sebuah ruang bawah tanah, dengan desain *level* prosedural. Pemain dituntut untuk mengoleksi barang tertentu, untuk bisa menyelesaikan *level* tersebut.

Gameplay pada *game* ini cukup sederhana. Pemain hanya diminta untuk menjelajahi *level* berperspektif 2D, menemukan gem, berpindah antar platform, serta menghindari serangan musuh. Karakter juga didesain memiliki kemampuan lompatan ganda untuk memudahkan pemain melewati dinding yang sangat tinggi atau terlalu jauh untuk dilewati pemain. Pemain juga bisa menginjak musuh untuk mengalahkan mereka.

Level yang dibuat pada pengembangan Dungeon Diver dibuat secara otomatis menggunakan algoritma C*ellular automata*. Hasil *level* yang dibentuk dengan aturan ini memiliki *layout* berbentuk kotak, mirip dengan *layout level platformer* maupun *sidescroller* umum seperti pada *game* Metroid, Megaman, dan lain-lain. Gambar 5 menunjukkan bentuk *layout level* yang dimaksud.



Gambar 6. Contoh *layout* dari *game* Metroid Fusion.

a. Storyboard

Struktur pada *game* ini sangat sederhana, seperti ditunjukkan pada Tabel 2 berikut:

Tabel 2. *Storyboard*

No.	Tampilan	Tombol	Audio
1	Menu utama	*Play* (*Scene* permainan dipilih secara acak), Keluar	*Background music*
2	scene permainan	*Pause*	*Background music*, *sfx* pemain dan musuh
3	*Pause*	Keluar, lanjutkan, menu utama	-
4	*Game Win*	*Retry* (kembali ke *scene* permainan), keluar	-
5	*Game Over*	*Retry* (kembali ke *scene* permainan), keluar	-

b. Kebutuhan sistem

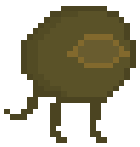
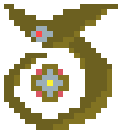
Storage yang dibutuhkan pada implementasi PCG ini memiliki kapasitas sekitar 400 mB, *RAM* 8 gB, *Quad core CPU* dengan kecepatan 2.6 GHz, dan kartu grafis *Radeon Graphics*.

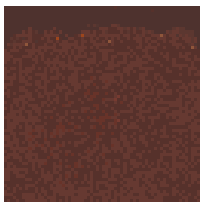
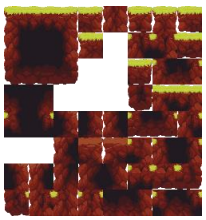
C. Material Collecting

Tahap *Material collecting* ini adalah tahap pengumpulan aset atau bahan-bahan yang diperlukan dalam pengembangan *game*, baik itu objek, suara dan musik, font untuk *User Interface* (UI), serta gambar latar belakang.

Beberapa objek yang diperlukan bisa dilihat pada Tabel 3 berikut ini:

Tabel 3. Objek *game*

Objek	Keterangan
	*Legged Slime*. Karakter yang pemain kendalikan dalam *game*. Merupakan makhluk sejenis slime berkaki dan berekor yang dapat melompat dua kali untuk melakukan lompatan yang jauh.
	*Eagle*. Musuh elang, salah satu musuh yang ada di dalam *game*. tetap pada satu tempat, namun terus menerus menembakkan proyektil ke arah pemain.
	*Reptile*. Salah satu musuh yang ada di dalam *game*. Sifat nya hampir mirip dengan elang, namun dapat menembakkan proyektil 2 kali lebih sering dibandingkan dengan elang.
	*Frog*. Salah satu musuh yang ada di dalam *game*. Tidak bisa menembakkan proyektil, namun terus mengikuti player.
	Ikon *hit point.*
	Gem. Ikon barang koleksi.

Objek	Keterangan
	Salah satu tile *environment* (lingkungan) yang dipakai untuk latar belakang.
	*Tile set* (Kumpulan tile) yang nantinya menjadi pijakan untuk Pemain.
	*Sprite* untuk judul.

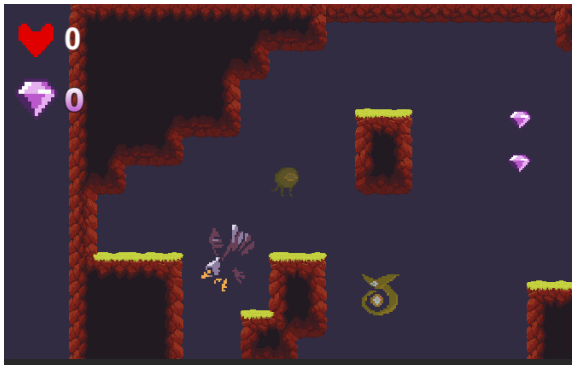
Sebagian dari aset yang dipakai dibuat secara pribadi, seperti objek gambar dan animasi, menggunakan aplikasi Aseprite. *Sound effect* (Sfx) atau efek suara dan musikya sendiri, menggunakan aset-aset gratis yang tersedia gratis di internet. Sfx yang digunakan diantaranya: sfx ledakan untuk musuh yang dikalahkan, sfx karakter melompat, sfx proyektil klasik, serta sfx mengoleksi barang. Untuk musiknya sendiri menggunakan beberapa musik *background* yang berbeda-beda.

D. Assembly

Tahap Assembly ini merupakan tahap pembuatan *game*. Tahap ini juga merupakan tahap perakitan dari berbagai objek yang telah dikumpulkan, berdasarkan desain yang telah direncanakan sebelumnya. Hasil perakitan aset *game* bisa dilihat pada Gambar 7 sampai dengan gambar 12 di bawah:

Gambar 7. Menu Utama

Gambar 7 menunjukkan sebuah layar *main menu*. Komponen yang dibuat sangat sederhana, terdiri dari Judul *game*, ikon karakter pemain, dan tombol *Play* dan *Quit*. Untuk memulai permainan, nantinya pemain cukup menekan tombol *Play* tersebut, kemudian pemain bisa memulai permainan dengan level yang akan dipilih secara acak



Gambar 8. *Level* Permainan

Tampilan permainan pada Gambar 8 menunjukkan sebuah level yang terdiri dari platform, dan 2 musuh yang dekat dengan karakter pemain, dan 2 buah gem di samping kanan. Tampilan UI yang ada menunjukkan indikator nyawa pemain, serta jumlah gem yang sudah terkumpul.

Gambar 9. *Pause*

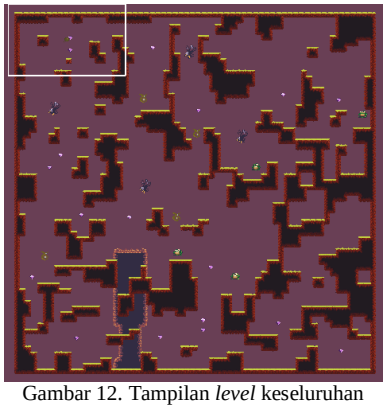
Gambar 9 menunjukkan tampilan *menu pause*, terdapat tombol *Main menu* untuk kemabali ke layar menu utama, serta *Quit* untuk menutup *game.*

Gambar 10. Layar Menang

Tampilan pada Gambar 10 merupakan tampilan layar ketika pemain berhasil mengumpulkan gem yang dibutuhkan untuk menang. Pada layar ini, pemain bisa memulai ulang permainan dengan menekan tombol *Retry*, atau langsung menutup permainan dengan tombol *Quit*

Gambar 11. Layar Kalah

Tampilan pada Gambar 11 merupakan layar ketika karakter pemain kehabisan jumlah nyawa. Tampilan layar kalah memiliki tombol yang sama dengan Layar Menang.



Gambar 12. Tampilan *level* keseluruhan

Gambar 12 menunjukkan hasil *level* yang telah dibuat setalah diperbesar. Dari hasil pembuatan *level* tersebut bisa dilihat bahwa tata letak *level* yang ada berbentuk kotak.

E. Testing

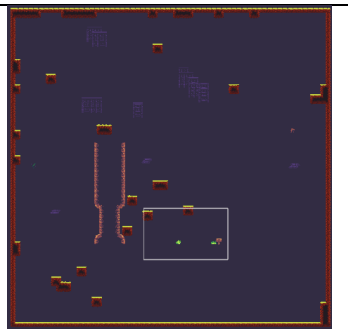
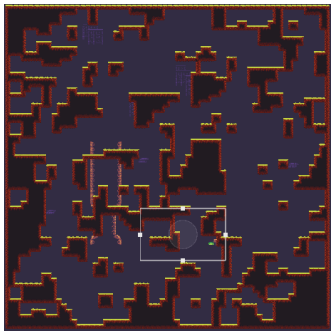
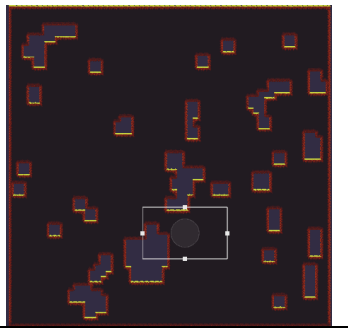
Pada tahap ini, dilakukan pengujian performa pembangkitan *level* pada *tile* berukuran 32x32 serta 64x64. Parameter yang diuji yaitu dengan *Fill percentage* serta *smooth amount*, dengan menggunakan *seed* acak. Hasilnya bisa dilihat pada Tabel 4 di bawah ini.

Tabel 4. Pengujian Performa

No.	*Fill percentage (%)*	*Smooth Amount*	Rata-rata waktu pada *tile* 32x32 (detik)	Rata-rata waktu pada *tile* 64x64 (detik)
1	25	2	0.0853	0.1128
2	25	4	0.0863	0.0896
3	25	6	0.0935	0.1139
4	45	2	0.0991	0.1786
5	45	4	0.1003	0.1767
6	45	6	0.0969	0.1932
7	65	2	0.1182	0.2761
8	65	4	0.1244	0.2776
9	65	6	0.1284	0.2755

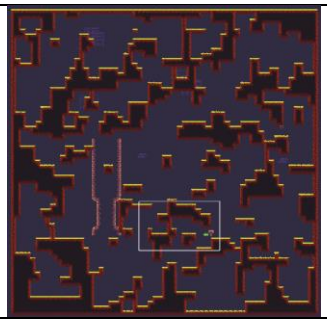
Pada pengujian fungsional, dilakukan pengujian untuk melihat apakah sebuah *level* dapat diakses oleh pemain, serta memiliki ruang kosong yang saling terhubung ataupun area untuk pemain lewati. Pengujian ini dilakukan dengan *fill percentage* 25%, 45%, serta 65% dengan *seed* yang sama. Hasilnya bisa dilihat pada Tabel 7 di bawah ini:

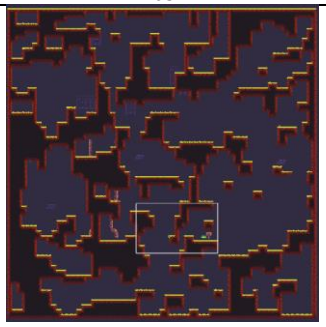
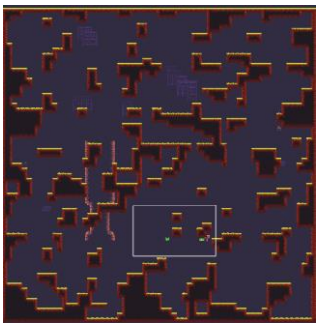
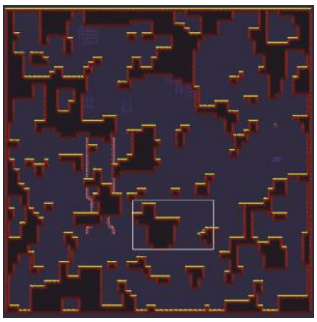
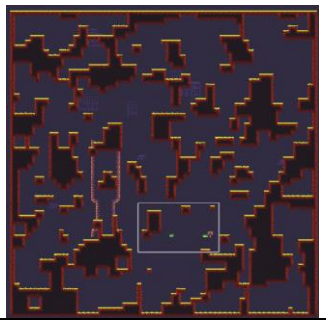
Tabel 5. Pengujian Fungsional

No.	*Fill percentage*	Hasil
1	25	
2	45	
3	65	

Perbandingan *layout* pada *fill percentage* 45% dengan *seed* yang berbeda, dapat dilihat pada Tabel 6 di bawah ini:

Tabel 6. Hasil *fill percentage* 45%.

No.	Seed	Hasil
1	1.8	

No.	Seed	Hasil
2	2	
3	12	
4	1232	
5	3435	

F. Distribution

Tahap terakhir dari MDLC adalah distribusi. Hasil keluaran dari penelitian ini berupa sebuah *game* berplatform PC Windows, yang didistribusikan dan bisa diunduh pada halaman website https://n0obguy.itch.io/dungeon-diver.

3.4. Implementasi

Level yang dibuat dibangun menggunakan sebuah *level generator* atau pembangkit, dimana *level generator* tersebut memuat referensi pada fungsi *Map Setting*, pengaturan ukuran dan seberapa padat *level* yang hendak dibuat, serta *gameObject* yang menjadi *tiling* atau *platform* dimana karakter berpijak. Pembangkit tersebut bisa membuat suatu *level* hanya dengan satu klik saja.

Skrip kode yang digunakan pada penerapan PCG *Cellular Automata* ini adalah sebagai berikut:

Skrip *load random level*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class LoadRandomScene : MonoBehaviour
{
public void LoadRandomScenes()
{
int index = Random.Range(1,10);
SceneManager.LoadScene(index); } }
```

Skrip *Cellular Automata*

```
public static int[,] GenerateCellularAutomata(int width, int height, float seed, int fillPercent, bool edgesAreWalls)
{
System.Random rand = new System.Random(seed.GetHashCode());
int[,] map = new int[width, height];
for (int x = 0; x < map.GetUpperBound(0); x++)
{
for (int y = 0; y < map.GetUpperBound(1); y++)
{
if (edgesAreWalls && (x == 0 || x == map.GetUpperBound(0) - 1 || y == 0 || y == map.GetUpperBound(1) - 1))
{
map[x, y] = 1; }
else
{
map[x, y] = (rand.Next(0, 100) < fillPercent) ? 1 : 0; } } }
return map; }
```

Aturan von neumann

```
static int GetVNSurroundingTiles(int[,] map, int x, int y, bool edgesAreWalls)
{
int tileCount = 0;
if(edgesAreWalls && (x - 1 == 0 || x + 1 == map.GetUpperBound(0) || y - 1 == 0 || y + 1 == map.GetUpperBound(1)))
{
tileCount++; }
if(x - 1 > 0)
{
tileCount += map[x - 1, y]; }
if(y - 1 > 0)
{
tileCount += map[x, y - 1]; }
if(x + 1 < map.GetUpperBound(0))
{
tileCount += map[x + 1, y]; }
```

```
if(y + 1 < map.GetUpperBound(1))
{
tileCount += map[x, y + 1]; }
        return tileCount; }
```

Skrip awal memiliki fungsi untuk memuat *level* secara acak dari index atau *list* kumpulan *level* yang sudah dibuat. Skrip kedua berfungsi untuk membuat dasar atau *Base grid* pembangkit *Cellular automata*, yang saling berkaitan dengan skrip berikutnya, berfungsi untuk membangkitkan *level* berdasarkan aturan *Von Neumann Neighborhood*. Metode yang saling terhubung pada kedua skrip tersebut adalah fungsi *GenerateCellularAutomata* dan fungsi *GetVNSurroundingTiles*.

Fungsi *GenerateCellularAutomata* bertanggung jawab sebagai pembangkit *level* berdasarkan parameter yang ada, seperti tinggi, lebar, *fill percentage*, serta mengecek tepi *level* dianggap dinding atau bukan dinding. Fungsi *GenerateCellularAutomata*, *level* diisi dengan nilai-nilai 0 dan 1 berdasarkan *fill percentage*, jika tepi peta dianggap dinding, maka tepi peta diisi dengan *tile* sehingga membentuk dinding. Fungsi *GetVNSurroundingTiles* menghitung jumlah sel tetangga yang valid dalam pola *Von Neumann Neighborhood* untuk sel tertentu dalam peta. Penghitungan validnya pola tetangga yang terbentuk pada *level*, dipengaruhi pada nilai parameter *edgesAreWalls*, berdasarkan nilai *boolean true* maupun *false*.

3.5. Evaluasi

Dari hasil pengujian performa yang dilakukan, terlihat bahwa *fill percentage*, *smooth amount*, serta ukuran *tile* bisa mempengaruhi waktu pembangkitan, namun tidak memiliki perbedaan yang terlalu signifikan antara parameter yang diuji. Waktu pembangkitan yang dihasilkan selalu berada dibawah 1 detik, yaitu pada rentang waktu 0.08 detik hingga 0.3 detik tiap pembangkitan.

Pada hasil pengujian fungsionalitas, menunjukkan bahwa *fill percentage* 25% hampir tidak memiliki pijakan serta hanya memiliki satu ruang besar saja. Sedangkan pada *fill percentage* 45%, terbentuk sebuah area yang saling terhubung, cukup luas untuk dilewati, namun masih memiliki banyak pijakan dan daerah yang terisolasi dapat berkurang. Pengujian terakhir untuk *fill percentage* 65%, terdapat banyak ruang kosong kecil yang tidak bisa dilalui oleh karakter yang dikendalikan pemain.

4. DISKUSI

Dalam mendesain *level* pada sebuah *game*, biasanya desainer membutuhkan waktu yang lama dalam membuat *level* tersebut. Hal ini dikarenakan desainer harus meletakkan tiap *tile* pijakan maupun dinding satu persatu, yang membuat proses pembuatan *level* menjadi sangat lama. Hasil uji yang dilakukan dalam penelitian ini berhasil meng-implementasikan *Procedural Content Generation* (PCG) dalam pembangkitan *level*. Hasil uji ini juga menunjukkan efisiensi waktu dalam pembuatan *level*. Hal tersebut menunjukkan potensi PCG dalam menghemat waktu seorang desainer dalam proses perancangan *level*, sehingga para desainer tidak memerlukan lagi waktu yang lama saat membangun sebuah *level*.

Hasil uji coba fungsional yang telah dilakukan juga menunjukkan bahwa penggunaan *fill percentage* 45% dianggap bisa digunakan untuk membuat *level* pada *dungeon level*. Cukup memiliki ruang yang saling terhubung, dapat dilalui pemain, serta memiliki pijakan yang cukup. Hasil ini memberikan referensi terkait parameter *fill percentage* yang tepat untuk digunakan dalam pembangkitan *level*. Hal ini tidak mengindikasikan bahwa penggunaan *fill percentage* yang berbeda tidak dapat digunakan. Hanya saja, untuk tujuan desain *game platformer sidescroller* 2D, hasil dari *fill percentage* 45% sudah dapat dianggap tepat. Pengujian ini juga menunjukkan bahwa perbedaan *seed* pada *fill percentage* yang sama menunjukkan hasil *layout* yang berbeda, sehingga variasi *layout level* yang bisa dihasilkan menjadi tidak terbatas.

Penelitian sebelumnya terkait penerapan PCG pada pengembangan *game* sudah dilakukan [16] menggunakan metode Algoritma Genetika. Algoritma genetika dapat menghasilkan level yang optimal dari hasil iterasi dengan memilih kandidat terbaik. Berbeda dengan metode tersebut, implementasi metode *cellular automata* yang dilakukan pada penelitian ini tidak memerlukan banyak iterasi, sehingga para *developer* dan desainer *game* bisa membuat *level* tanpa menunggu proses yang lama.

5. KESIMPULAN

Proses pembuatan *level* menggunakan metode PCG memiliki tingkat efisiensi yang tinggi, dengan waktu pembuatan rata-rata berkisar antara 0.08 detik hingga 0.3 detik, yang berarti perancang *level game* dapat menghasilkan berbagai variasi *level* dengan cepat dan efisien.

Pengujian fungsional dengan menggunakan tiga variasi *fill percentage* yang berbeda menghasilkan bentuk level yang beragam. *Fill percentage* sebesar 25% menghasilkan ruang yang luas dengan sedikit pijakan, *fill percentage* 45% menciptakan *level* dengan keseimbangan antara ruang kosong dan pijakan, sementara *fill percentage* 65% menghasilkan *level* dengan sedikit ruang yang dapat dilewati. Selain itu, pengujian dengan menggunakan *fill percentage* yang sama yaitu 45% dengan *seed* yang berbeda, berhasil menciptakan *layout level* yang berbeda satu sama lain.

Berdasarkan hal tersebut, dapat disimpulkan PCG merupakan metode yang efektif dalam menciptakan *level game* dengan cepat dan variasi

yang beragam. PCG dapat meningkatkan efisiensi dalam proses perancangan *level* dengan berbagai tingkat kesulitan dan struktur. Dengan variasi yang dihasilkan oleh parameter *fill percentage* dan *seed* yang berbeda, PCG juga memungkinkan untuk menghasilkan level yang unik dan beragam dalam *game*, meningkatkan pengalaman pemain dan mendukung kreativitas dalam pembuatan *game*.

DAFTAR PUSTAKA

[1] T. Stalnaker, "Procedural Generation of Metroidvania Style Levels," Washington and Lee University, 2020. [Online]. Available: https://dspace.wlu.edu/bitstream/handle/11021/34738/WLURG38_Stalnaker_CSCI_2020.pdf

[2] A. B. Moghadam and M. K. Rafsanjani, "A genetic approach in procedural content generation for platformer games level creation," *2nd Conf. Swarm Intell. Evol. Comput. CSIEC 2017 - Proc.*, no. February, pp. 141–146, 2017, doi: 10.1109/CSIEC.2017.7940160.

[3] S. Risi and J. Togelius, "Increasing generality in machine learning through procedural content generation," *Nat. Mach. Intell.*, vol. 2, no. 8, pp. 428–436, 2020, doi: 10.1038/s42256-020-0208-z.

[4] A. M. Abuzuraiq, O. Alsalman, and H. Erhan, "Shopping for Game levels: A Visual Analytics Approach to Exploring Procedurally Generated Content," *ACM Int. Conf. Proceeding Ser.*, pp. 2–5, 2020, doi: 10.1145/3402942.3409793.

[5] B. M. F. Viana and S. R. Dos Santos, "Procedural Dungeon Generation: A Survey," *J. Interact. Syst.*, vol. 12, no. 1, pp. 83–101, 2021, doi: 10.5753/jis.2021.999.

[6] J. Liu, S. Snodgrass, A. Khalifa, S. Risi, G. N. Yannakakis, and J. Togelius, "Deep learning for procedural content generation," *Neural Comput. Appl.*, vol. 33, no. 1, pp. 19–37, 2021, doi: 10.1007/s00521-020-05383-8.

[7] N. Muir and S. James, "Combining Evolutionary Search with Behaviour Cloning for Procedurally Generated Content," *Epic Ser. Comput.*, vol. 85, pp. 77–88, 2022, doi: 10.29007/qpkt.

[8] C. S. Putri, E. M. A. Jonemaro, and M. A. Akbar, "Penerapan Procedural Content Generation pada Pembangkit Level Gim Maze Heksagonal," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 3, no. 9, pp. 8563–8571, 2019, [Online]. Available: http://j-ptiik.ub.ac.id

[9] C. Miller, M. Dighe, C. Martens, and A. Jhala, "Stories of the town: Balancing character autonomy and coherent narrative in procedurally generated worlds," *ACM Int. Conf. Proceeding Ser.*, 2019, doi: 10.1145/3337722.3341850.

[10] M. A. Muslim, E. M. A. Jonemaro, and M. A. Akbar, "Penerapan Procedural Content Generation untuk Perancangan Level pada 2D Endless Runner Game menggunakan Genetic Algorithm," *Garuda - Garba Rujukan Digit.*, vol. 3, no. 5, pp. 4406–4414, 2019, [Online]. Available: https://garuda.ristekbrin.go.id/documents/detail/992300

[11] R. C. E. Silva, N. Fachada, N. Códices, and D. De Andrade, "Procedural Game Level Generation by Joining Geometry with Hand-Placed Connectors," pp. 1–14, 2020.

[12] A. Gellel and P. Sweetser, "A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels," *ACM Int. Conf. Proceeding Ser.*, 2020, doi: 10.1145/3402942.3402945.

[13] H. Mohr, M. Eger, and C. Martens, "Eliminating the impossible: A procedurally generated murder mystery," *CEUR Workshop Proc.*, vol. 2282, 2018.

[14] N. C. Rikandi and S. R. Nudin, "Rancang Bangun Visual Novel Peduli Lingkungan dengan Metode Procedural Content Generation," *J. Informatics Comput. Sci.*, vol. 4, no. 01, pp. 131–142, 2022, doi: 10.26740/jinacs.v4n01.p131-142.

[15] S. Tolinsson, A. Flodhag, A. Alvarez, and J. Font, "To make sense of procedurally generated dungeons," *CHI Play 2020 - Ext. Abstr. 2020 Annu. Symp. Comput. Interact. Play*, pp. 384–387, 2020, doi: 10.1145/3383668.3419890.

[16] D. Wijaya, H. Haryanto, E. Z. Astuti, and W. Wijanarto, "Algoritme Genetika untuk Desain Level Dinamis pada Game Edukasi Kebakaran Hutan," *Komputika J. Sist. Komput.*, vol. 10, no. 1, pp. 69–76, 2021, doi: 10.34010/komputika.v10i1.3586.

[17] R. I. Borman and Y. Purwanto, "Implementasi Multimedia Development Life Cycle pada Pengembangan Game Edukasi," *J. Edukasi dan Penelit. Inform.*, vol. 5, no. 2, pp. 119–124, 2019.