Infotekmesin
Vol.15, No.02, Juli 2024
p-ISSN: 2087-1627, e-ISSN: 2685-9858
DOI: 10.35970/infotekmesin.v15i2.2317, pp. 321-329

Implementasi Algoritma *Random walk* untuk *Procedural generation* dalam Genre *Game Dungeon Crawler* 2D

Dani Arifudin[1*], Dhipa Ringi Rizky[2], Andi Dwi Riyanto[3], Suliswaningsih[4]
[1, 2]Teknologi Informasi, Universitas Amikom Purwokerto
[3]Sistem Informasi, Universitas Amikom Purwokerto
[4]Informatika, Universitas Amikom Purwokerto
[1,2,3,4]Jln. Letjend Pol, Soemarto No.127 Watumas, Kel.Purwanegara, Kec. Purwokerto Utara, Kab. Banyumas, Indonesia
E-mail: daniarif@amikompurwokerto.ac.id[1], dhipa.r.rizky@amikompurwokerto.ac.id[2], andi@amikompurwokerto.ac.id[3], suliswani@amikompurwokerto.ac.id[4]

Info Naskah:
Naskah masuk: 25 Mei 2024
Direvisi: 4 Juli 2024
Diterima: 11 Juli 2024

Abstrak

Pemanfaatan algoritma untuk menghasilkan konten melalui *procedural generation* menjadi alternatif yang menarik dalam pengembangan *game*. Namun, perlu diperhatikan bahwa terdapat berbagai jenis algoritma dengan hasil yang beragam, serta kesesuainnya terhadap jenis *game* 2D/3D. *Random walk* adalah algoritma sederhana yang sering digunakan dalam *procedural generation*. Algoritma ini bekerja pada ruang diskrit atau kontinu, dari posisi awal dan bergerak dengan mengambil arah acak pada setiap langkah. *Genre game Dungeon Crawler* adalah jenis *game* yang dapat memanfaatkan *procedural generation*. Genre *game* yang berkembang dari genre RPG, berfokus pada eksplorasi labirin atau ruang bawah tanah yang sering kali diacak dan penuh dengan berbagai rintangan. Genre ini menawarkan sejumlah fitur menarik, seperti mengumpulkan hadiah explorasi, menyelesaikan teka-teki dan mencari jalan keluar. Penelitian ini menggunakan GDLC sebagai metode pengembangan system berbasis *device Windows* PC. Hasil pengujian dengan menggunakan metode *Black Box* dapat dikatakan sukses karena *game* berhasil menciptakan level secara otomatis dengan mengimplementasikan algoritma *Random walk*.

Keywords:
random walk,
procedural generation,
GDLC,
game 2D,
dungeon crawler

Abstract

The use of algorithms to produce content through procedural generation is an attractive alternative in game development. However, it should be noted that there are various types of algorithms with varying results, as well as their suitability for 2D/3D game types. Random walk is a simple algorithm that is often used in procedural generation. This algorithm works on discrete or continuous space, from an initial position and moves by taking a random direction at each step. The Dungeon Crawler game genre is a type of game that can utilize procedural generation. A game genre that developed from the RPG genre, focusing on the exploration of mazes or dungeons that are often randomized and full of various obstacles. This genre offers several interesting features, such as collecting exploration rewards, solving puzzles, and finding a way out. This research uses GDLC as a system development method based on Windows PC devices. The test results using the Black Box method can be said to be successful because the game succeeded in creating levels automatically by implementing the Random walk algorithm.

***Penulis korespondensi:**
Dani Arifudin
E-mail: daniarif@amikompurwokerto.ac.id

1. Pendahuluan

Game 2D masih diminati oleh sejumlah besar pemain, baik yang bermain di konsol maupun perangkat seluler. *Game* 2D menawarkan keunggulan tertentu yang membuatnya menarik bagi berbagai kalangan. Mengembangkan *game* 2D lebih mudah karena berbagai elemen memiliki tingkat kompleksitas yang lebih rendah dibanding pengembangan *game* 3D, sehingga tidak memerlukan banyak sumber daya, dan tenaga kerja. Keunggulan lainnya adalah *game* 2D lebih mudah dipahami dan dipelajari oleh pemain karena memiliki lebih sedikit elemen yang perlu dipelajari, bahkan pemula yang tidak memiliki pengalaman bermain *game* [1].

Pengembangan *game* telah menjadi semakin kompleks dan menantang seiring dengan pertumbuhan industri *game* dan meningkatnya permintaan konten *game* yang lebih menarik dan inovatif. Namun, pembuatan konten *game* secara manual membutuhkan sumber daya yang signifikan. Pengembangan *game* menghadapi tantangan seperti menciptakan desain lingkungan *game* yang kredibel dalam waktu yang terbatas, serta menjaga agar proyek tetap berjalan sesuai anggaran sehingga hasilnya tetap dapat dijangkau oleh konsumen [2]. Untuk mengatasi masalah ini, penggunaan *algoritma* untuk menghasilkan konten dapat menjadi solusi yang efektif dalam menghemat waktu dan biaya.

Prosedur pembuatan konten secara algoritmik ini disebut *procedural generation*. Secara spesifik, penggunaan metode yang sama dalam konteks *game* dan aplikasi disebut *Procedural Content Generation* (PCG). Namun, kedua istilah tersebut dapat digunakan secara bergantian. *Procedural Content Generation* (PCG) merupakan metode yang diterapkan dalam proses pengembangan permainan dan komputasi grafis dengan tujuan menghasilkan konten permainan secara otomatis melalui penerapan *algoritma* atau peraturan matematis. Konten permainan yang dapat dihasilkan melalui PCG mencakup beragam elemen, seperti level permainan, peta, karakter, senjata, misi, tekstur, dan elemen lainnya. Prinsip utama dari PCG adalah meningkatkan keragaman, daya ulang, dan efisiensi dalam proses pengembangan permainan, sambil memberikan pengalaman bermain yang unik setiap kali permainan dimainkan [3]. Beberapa *algoritma* ini adalah seperti *algoritma* Genetika, *Perlin Noise*, *Cellular Automata*, *Recursive Backtracking*, dan *Random walk*.

Penggunaan *algoritma* memiliki daya tarik tersendiri dalam *procedural generation*, karena sifatnya yang beragam dan mampu menyajikan hasil yang berbeda, terutama ketika diaplikasikan dalam genre *game* tertentu. Sebagai contoh, *algoritma perlin noise* sering digunakan dalam pembuatan pemandangan atau lingkungan dalam *game* [4]. *Perlin noise* menghasilkan tekstur dengan gradasi halus yang memberikan kesan alami pada konten yang dihasilkan [5]. Hal ini membuatnya cocok untuk menciptakan lanskap, hutan, dan berbagai elemen lingkungan dalam *game* yang tampak realistis [6].

Studi berjudul "*Procedural Content Generation* pada *Game World Exploration Sandbox* Menggunakan *Algoritma Perlin Noise*" menyimpulkan penggunaan *Perlin noise* dalam *game* 2D dapat menghasilkan bentuk bukit dan gua yang natural, namun untuk area awan yang terbentuk tidak cocok untuk bentuk *game* 2 dimensi dengan sudut pandang perspektif *orthogonal view*. Pembentukan area awan lebih cocok untuk sudut pandang perspektif *isometric view* [7].

Kesimpulan bahwa pendekatan berdasarkan *noise-generator* seperti *perlin noise* mengalami kendala dalam mengelola medan yang dihasilkan. Meskipun sebagian memungkinkan batasan melalui parameter di awal, menciptakan elemen-elemen lingkungan yang sesuai dengan batasan yang sangat spesifik terbukti sulit dilakukan dengan efisien [8]. *Perlin noise* kurang cocok untuk *game* 2D karena kompleksitasnya sebagai *algoritma* jenis *noise-generator*, diperlukan *algoritma* lain yang lebih sederhana dan memiliki parameter yang lebih mudah diatur. Algoritma *random walk* adalah salah satu *algoritma* yang dapat memenuhi kebutuhan tersebut.

Random walk adalah *algoritma* sederhana yang sering digunakan dalam *procedural generation* untuk menciptakan pola acak yang tidak dapat diprediksi dan dapat menyerupai labirin [9]. Algoritma ini bekerja pada ruang diskrit atau kontinu, dimulai dari posisi awal dan bergerak dengan mengambil arah acak pada setiap langkah. Setiap langkah acak ditambahkan ke posisi saat ini untuk menentukan posisi selanjutnya, dan proses ini diulang dalam sejumlah langkah tertentu atau sampai mencapai kondisi berhenti [10].

Sejalan dengan Penelitian yang dilakukan oleh Ijai, dkk (2023), penelitian ini mengungkapkan bahwa penerapan teknik *Random walk* berhasil menghasilkan variasi peta yang dinamis di setiap sesi permainan, mengatasi kebosanan yang biasanya disebabkan oleh repetisi. Implikasi dari temuan ini adalah peningkatan replayability *game*, memungkinkan pemain menikmati pengalaman bermain yang lebih menarik dan dinamis karena mereka tidak akan menghadapi peta yang sama setiap kali memulai permainan. Dengan demikian, penggunaan Algoritma *Random walk* untuk menciptakan peta prosedural dapat memberikan kontribusi signifikan dalam meningkatkan pengalaman bermain dalam *game* [11]. Kemudian pada penelitian Pandey, S (2019) mengungkapkan efisiensi berbagai strategi pencarian dengan *random walk* dianalisis menggunakan graf acak sebagai ruang pencarian. Graf acak GGG didefinisikan oleh sekumpulan simpul VVV dan sekumpulan sisi EEE yang diwakili oleh matriks ketetanggaan CCC. Dengan memanfaatkan metode *cavity* untuk analisis instansi tunggal yang besar, penelitian ini memperluas pekerjaan sebelumnya untuk mencakup *random walk* yang bias derajat. Hasilnya menunjukkan bahwa strategi pencarian dengan *random walk* yang bias derajat efektif dalam menemukan item yang tersembunyi pada simpul graf acak, dengan efisiensi yang konsisten baik pada analisis teoretis maupun simulasi numerik [12].

Penentuan titik awal untuk memulai *random walk* akan dilakukan dari titik tengah objek yang telah di-*capture*. Titik awal ini diperoleh dengan menggunakan teknik *edge detection* pada pemrosesan gambar. Hasil ini akan digunakan untuk dipadukan dengan metode *random walk* yang melalui suatu *controller* akan menghasilkan

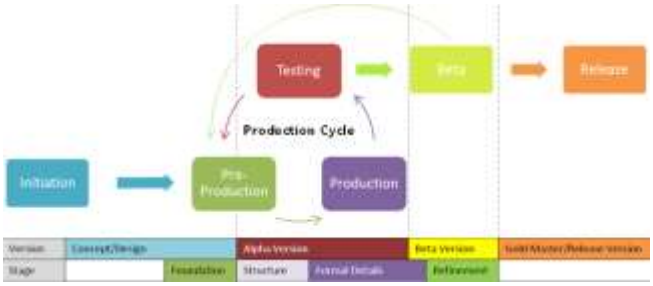
pergerakan hingga mencapai titik akhir, yaitu posisi yang tepat berada di tengah citra [13].

Dungeon crawler merupakan genre *game* yang berkembang dari genre RPG, berfokus pada eksplorasi labirin atau ruang bawah tanah yang sering kali diacak dan penuh dengan berbagai rintangan. Genre ini menawarkan sejumlah fitur menarik, seperti sistem pertarungan taktis dalam menghadapi musuh, mengumpulkan hadiah explorasi, menyelesaikan teka-teki dan mencari jalan keluar [14]. Beberapa contoh *game* dengan genre *Dungeon Crawler* adalah : *Moonlighter, seri Diablo, Path of Exile* [15].

Melihat dari permasalahan yang ada dilakukan penelitian mengenai permasalahan pembuatan konten pada pembuatan genre *game* Dungeon Crawler 2D khususnya pada pembuatan level secara *procedural generation* yang mengimplementasikan Algoritma *Random walk* dengan tujuan untuk memberikan metode alternatif dalam pengembangan *game*, yang dapat menciptakan level secara otomatis dan dinamis dengan judul "Implementasi Algoritma *Random walk* untuk *Procedural generation* dalam Genre *Game Dungeon Crawler* 2D".

2. Metode

Metode pengembangan GDLC (*Game Development Life Cycle*) merupakan proses iteratif yang terdiri dari 6 fase pengembangan utama, yaitu *Initiation* (Inisiasi), *Pre-production* (Pra-Produksi), *Production* (Produksi), *Testing* (Pengujian), *Release* (Perilisan) [16] . Pada Gambar 1 merupakan fase dan proses *Game Development Life Cycle*.



Gambar 1. Fase dan Proses GDLC

2.1 *Initiation* (Inisiasi)

Initiation adalah tahap awal yang melibatkan pembuatan konsep dan skenario dari *game* yang akan dikembangkan. Pada tahap ini, tim pengembang akan merumuskan skenario *game*, karakter, cerita dalam *game*, target pemain, *platform* yang akan digunakan, serta *game engine* yang akan menjadi basis pengembangan. Proses ini juga mencakup identifikasi tujuan utama *game*, menentukan genre, dan membuat perencanaan awal mengenai fitur-fitur unik yang ingin ditawarkan. Diskusi mendalam tentang visi dan misi *game* penting dilakukan untuk memastikan semua anggota tim memiliki pemahaman yang sama tentang arah pengembangan *game*

2.2 *Pre-production* (Pra-Produksi)

Pre-production adalah tahap di mana konsep desain *game* dan prototipe awal dibuat. Tahap ini melibatkan detail desain seperti mekanika *gameplay*, karakter, tantangan, dan aspek teknis. Setiap elemen *game* diuraikan secara rinci dalam *Game* Design Document (GDD), yang berfungsi sebagai panduan selama proses pengembangan. Prototipe awal, yang merupakan versi sederhana dari *game*, dibuat untuk menguji ide dan mekanika dasar. Diskusi dan peninjauan berkala dilakukan untuk memastikan desain sesuai dengan visi awal dan memungkinkan untuk direalisasikan dalam tahap produksi.

2.3 *Production* (Produksi)

Production adalah tahap di mana aset *game* seperti grafis, musik, suara, dan kode sumber dikembangkan dan diintegrasikan. Pada tahap ini, tim pengembang bekerja pada peningkatan elemen-elemen yang telah ditentukan pada tahap *Pre-production*. Pembuatan aset melibatkan berbagai disiplin ilmu, termasuk desain grafis, animasi, dan pemrograman. Selama tahap ini, komunikasi yang baik antar anggota tim sangat penting untuk memastikan semua elemen bekerja harmonis. Revisi dan iterasi dilakukan secara terus-menerus untuk mengatasi masalah yang muncul dan untuk meningkatkan kualitas *game*.

2.4 *Testing* (Pengujian)

Testing adalah tahap di mana *game* diuji secara internal untuk memastikan fungsionalitas dan kualitasnya. Proses pengujian melibatkan identifikasi dan perbaikan *bug* atau kesalahan, serta evaluasi fitur-fitur dalam *game* untuk menentukan apakah perlu ada penambahan atau pengurangan. Pengujian dilakukan dalam beberapa fase, mulai dari *alpha testing* hingga *beta testing*, dengan melibatkan *tester* internal dan eksternal untuk mendapatkan umpan balik yang beragam. Pengujian juga mencakup aspek-aspek seperti performa, kompatibilitas *platform*, dan pengalaman pengguna.

2.5 *Release* (Perilisan)

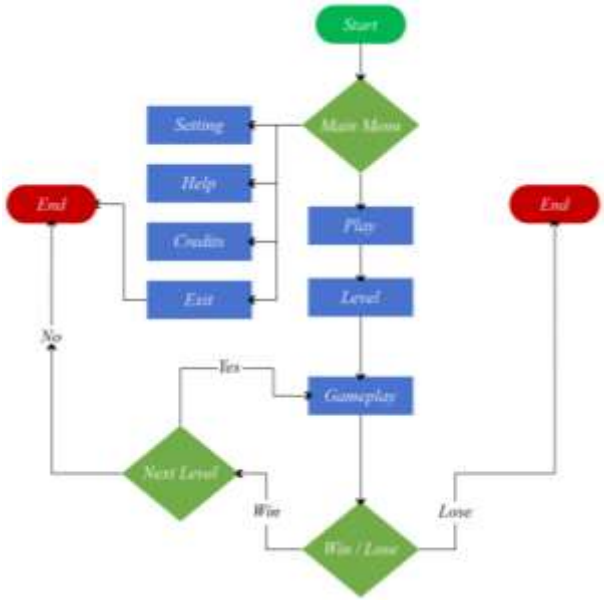
Setelah *game* dinyatakan lulus tahap pengujian, *game* siap untuk dirilis ke publik. Tahap ini melibatkan persiapan akhir seperti pembuatan materi promosi, penyiapan *server* untuk *game* online, dan koordinasi dengan platform distribusi seperti *Steam, App Store*, atau *Google Play*. Peluncuran *game* juga mencakup kegiatan pemasaran untuk menarik perhatian pemain potensial. Setelah perilisan, tim pengembang tetap harus memantau kinerja *game* dan siap untuk merilis *patch* atau *update* jika ditemukan masalah atau jika ada permintaan fitur dari komunitas pemain.

3. Hasil dan Pembahasan

3.1 *Inititation* (Inisiasi)

Konsep *game* yang akan dibuat adalah *game* dengan genre *Dungeon Crawler*. Pemain diharuskan menjelajahi labirin bawah tanah yang gelap, menghadapi musuh yang muncul, dan menemukan portal untuk melanjutkan ke level berikutnya. Setiap level dihasilkan secara acak, dan memiliki tata letak yang berbeda setiap kali dimainkan.

Berikut pada Gambar 2 merupakan alur *game Dungeon Crawler*.



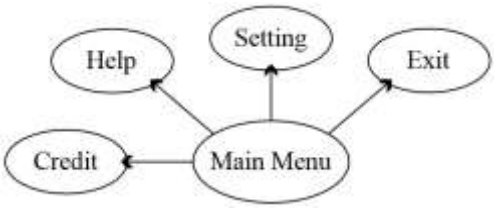
Gambar 2. Alur *Game Dungeon Crawler*

Gambar 2 menjelaskan alur *game Dungeon Crawler*. Bentuk interaktifitas *game* yaitu pemain memilih menu dan memainkan permainan. Pemain akan menjelajah *dungeon* dan mengalahkan musuh, setelah itu dapat melanjutkan ke *level* berikutnya dengan memasuki portal. *Level* akan berada dalam kondisi gelap dan pemain akan terus bermain hingga mengalami kekalahan. *Game* ini memiliki alur cerita yaitu seorang Ksatria jatuh ke ruang bawah tanah yang terkutuk, meskipun tidak ada kemungkinan untuk kembali ke permukaan, dia tetap akan terus maju dan bertahan hidup.

3.2 *Pre-production* (Pra-Produksi)

a) Perancangan

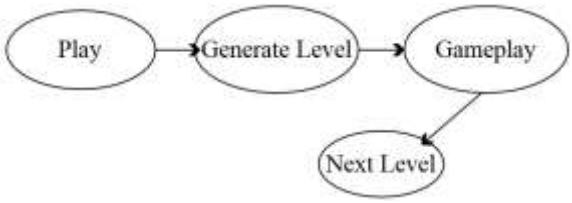
Pada Gambar 3 merupakan tampilan menu utama *game Dungeon Crawler* menampilkan judul *game* dan beberapa tombol pilihan yaitu *credit*, *help*, *setting*, dan *exit*. Adapun penjelasan fungsi masing-masing tombol dapat dilihat pada Tabel 1. Pada Gambar 4 merupakan tampilan *gameplay* pada *game Dungeon Crawler* menampilkan beberapa tombol pilihan yaitu *play*. Adapun penjelasan fungsi masing-masing tombol dapat dilihat pada Tabel 2. Pada Gambar 5 merupakan tampilan *gameplay* pada *game Dungeon Crawler* menampilkan tombol pilihan yaitu *exit*, *setting*, dan *resume*. Adapun penjelasan fungsi masing-masing tombol dapat dilihat pada Tabel 3.



Gambar 3. *Use Case - Main Menu*

Tabel 1. Deskripsi *Use Case - Main Menu*

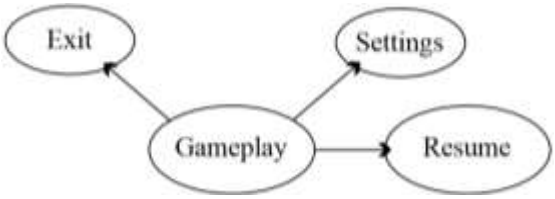
Use Case	*Main Menu*	
Aktor	*Player*	
Deskripsi	Proses ketika *player* membuka *game*.	
Kondisi	**Player**	***Game***
1.	Membuka *game*	Menampilkan *main menu*
2.	Memilih *settings*	Menampilkan halaman *settings* untuk mengatur resolusi layar *game* dan *audio*
3.	Memilih *help*	Menampilkan panel *help yang* memberi penjelasan elemen-elemen *game*, termasuk kontrol *player*.
4.	Memilih *credit*	Menampilkan panel *credits* yang berisi informasi tentang pengembang dan aset pendukung dalam *game*
5.	Memilih *exit*	Keluar dari *game.*



Gambar 4. *Use Case – Play*

Tabel 2 Deskripsi *Use Case - Play*

Use Case	*Play*	
Aktor	*Player*	
Deskripsi	Proses ketika *player* memilih *Play*	
Kondisi	**Player**	***Game***
1.	Menekan tombol *Play*	*Game* memuat *level* dengan *algoritma random walk*. *enemy*, *item*, dan portal untuk menuju *level* selanjutnya dimunculkan didalam batasan *level* yang sudah ditentukan dalam *algoritma* tersebut.



Gambar 5. *Use Case – Gameplay*

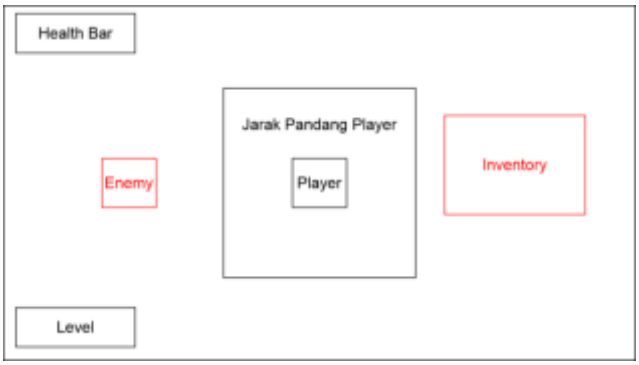
Tabel 3 Deskripsi *Use Case - Gameplay*

Use Case	*Gameplay*	
Aktor	*Player*	
Deskripsi	Proses Saat *player* berada di dalam permainan	
Kondisi	**Player**	***Game***
1.	Mengambil *item*	Memasukkan *item* ke dalam *inventory*
2.	Mengalahkan *Enemy*	*Enemy* dihilangkan dari level
3.	Memasuki	*Level* selanjutnya dimuat secara,

	portal	*player*, *enemy* dan *item* diletakan secara acak.
4.	Menekan tombol "*Esc*"	*Game* akan terjeda dan menampilkan pilihan tombol *Resume*, *Settings*, *Exit*
5.	Memilih tombol saat *game* terjeda	Tombol *"Resume"* akan menutup pillihan dan melanjutkan *gameplay*. Tombol "*Setting"* akan membuka panel *settings*. Tombol *"Exit"* akan mengembalikan *player* ke *Main Menu*.

b) *Gameplay*

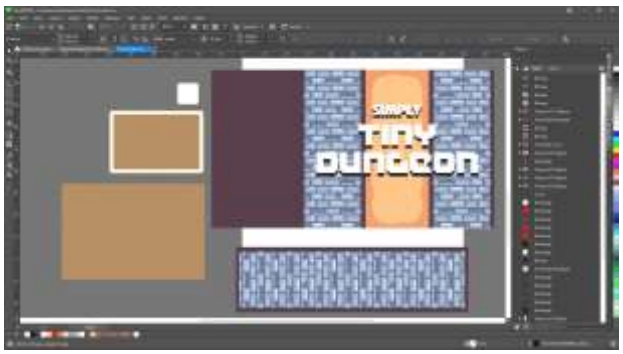
Gameplay untuk *game Dungeon Crawler* memiliki beberapa informasi penting, seperti *level* yang sedang dimainkan (*Floor*), dan *health bar* pemain. Pemain dapat bergerak dengan menekan tombol arah, atau tombol "W, A, S, D" mengakses *inventory* dengan menekan tombol "I", berinteraksi dengan peti, barel, *green potion* menggunakan tombol "F", menyerang musuh dengan tombol "J", dan menjeda *game* dengan menekan tombol "*Esc*" untuk memunculkan *pause menu*. Gambar 6 adalah desain tampilan *gameplay Dungeon Crawler*.



Gambar 6. *Gameplay Dungeon Crawler*

c) *User Interface* (UI)

Pembuatan UI (*User Interface*) adalah proses mendesain dan mengembangkan antarmuka untuk sebuah *software*, yang mencakup berbagai elemen visual, seperti tombol, ikon, dan menu. Gambar 7 dan Gambar 8 merupakan contoh aset berupa visual yang dibuat menggunakan *software* desain *Corel* dan *Pixelorama.*

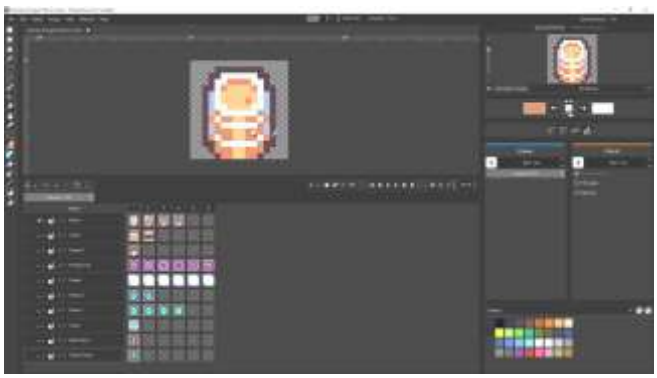


Gambar 7 Contoh pembuatan UI dan *Game Title*

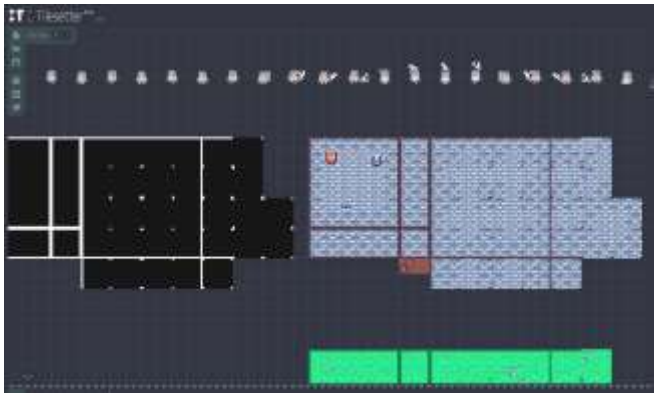
Gambar 8. Contoh pembuatan UI tombol

d) *Tileset*

Pembuatan *tileset* adalah proses membuat kumpulan gambar yang dapat digunakan untuk membuat dalam *game*. *Tileset* terdiri dari gambar yang mencakup berbagai aspek seperti lingkungan dan karakter dalam *game*, berukuran sama, dan dapat digabungkan untuk membentuk berbagai pola. Gambar 9 dan Gambar 10 merupakan contoh aset *tileset* yang dibuat menggunakan *software Pixelorama* dan *Tilesetter*.



Gambar 9. Contoh pembuatan *tileset*



Gambar 10. Contoh penyusunan *tileset* menggunakan *tilesetter*

e) *Audio*

Audio juga merupakan *asset* dalam *game* menggunakan *software* pengolah audio yaitu *Chiptone*. Pengolahan audio untuk *game Dungeon Crawler* ditujukan seperti pada Gambar 11.

Gambar 11. Pembuatan audio menggunakan *Chiptone*

3.3 *Production* (Produksi)

a. *User Interface* (UI)

Dalam pembuatan UI, hal yang penting adalah mendesain secara konsisten, serta sesuai dengan *genre* dan target audiens *game*, agar pemain dapat dengan mudah menggunakan dan memahaminya. Pada Gambar 12 dan adalah desain *Main Menu* menggunakan latar belakang dari aset *tileset level*, teks judul besar, dan dekorasi berupa obor menyala untuk menciptakan kesan hidup dan menarik perhatian *player*. Sedangkan Gambar 13 menunjukkan berbagai aset seperti tombol, *panel*, dan simbol digabungkan untuk menghasilkan tampilan menarik dan nyaman di mata pemain.

Gambar 12. Tampilan UI *Main Menu*

Gambar 13. Tampilan UI *Help Panel*

b. *Player*

Pada pembuatan *Player* di *Godot Editor* dibuat dengan menggabungkan kode, dan serangkaian animasi sesuai *input* yang mengatur pergerakannya menjadi *state machine*. *Enemy* diam selama 3 detik, lalu bergerak ke arah acak, lalu diam lagi, dan mengulangi siklus ini. *Player* memiliki 3 area deteksi: untuk mendeteksi tembok dan objek lain (*collision*), untuk mendeteksi *enemy* dalam jangkauan serangan (*hitbox*), dan untuk menangkap input serangan dari *enemy* (*hurtbox*). Jika *player* memasuki area deteksi kedua, maka *enemy* akan mengejar *player*. *Player* memancarkan cahaya terang, namun tidak terlalu luas.

Kode program pada *player* memiliki beberapa variabel seperti *animationTree* untuk mengatur animasi sesuai input pada fungsi *move_state*, terdapat juga variabel *inventory* yang terhubung dengan fungsi *collect* untuk memasukkan item yang diambil *player* kedalam *inventory*.

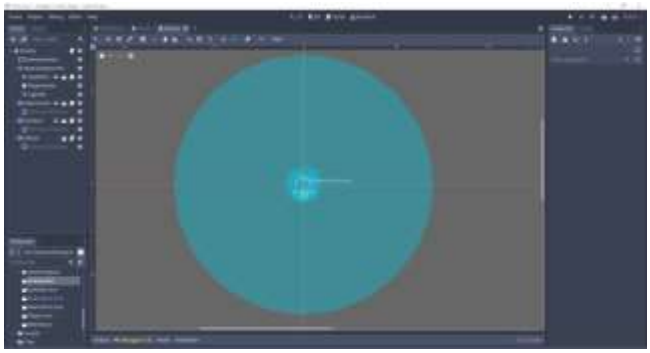
Gambar 14. pembuatan *Player* pada *Godot Editor*

Gambar 14 terlihat *collision* dan *hurbox* milik *player* berwarna biru, *hitbox* warna putih yang aktif saat *player* menyerang, dan *Health bar* milik *player* yang berwarna hijau.

c. *Enemy*

Pada pembuatan *Enemy* di *Godot Editor* dilakukan dengan menggabungkan kode yang mengatur pergerakan dengan *state machine* sederhana. *Enemy* diam selama 3 detik, lalu bergerak ke arah acak, lalu kembali diam, dan mengulangi siklus ini. *Enemy* memiliki 3 area deteksi: untuk mendeteksi tembok dan objek lain (*collision*), untuk mendeteksi *player* dari jarak tertentu, dan untuk menangkap *input* serangan dari *player (hurtbox)*. Jika *player* memasuki area deteksi kedua, maka *enemy* akan mengejar *player*. Apabila menyentuh, maka *health* milik *player* akan berkurang. *Enemy* juga memancarkan cahaya berwarna merah yang redup. Kode program pada *enemy* memiliki beberapa variabel seperti *detection_area* untuk mengatur area deteksi agar *enemy* mengejar jika *player* masuk kedalamnya, dalam fungsi *seek_player*. Terdapat juga variabel *knockback_velocity* yang terhubung dengan fungsi *_on_Hurtbox_area_entered* yang berfungsi merespon serangan *player* dengan terpental mundur.

Gambar 15 terlihat *collision* dan area deteksi berukuran besar milik *enemy*, keduanya berwarna biru dengan ukuran yang berbeda.



Gambar 15. pembuatan *enemy* pada *Godot Editor*

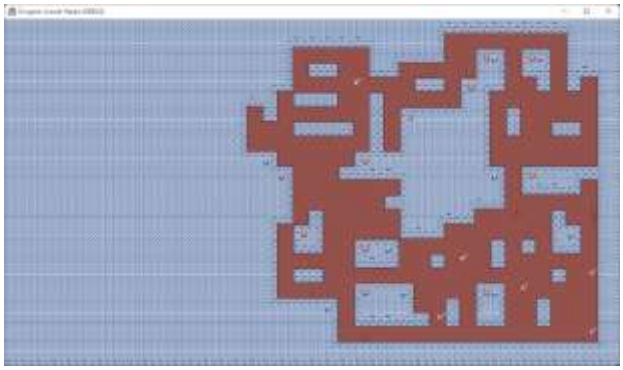
d. Pembuatan *level* dengan *algoritma Random walk*

Gambar 16 merupakan kode *random walk* berawal dari variabel direction yang menentukan arah *algoritma* bergerak, *step_history* untuk menyimpan variabel arah saat melangkah, *step_since_turn* untuk menghitung langkah. Pada kode berikut, fungsi *_init* melakukan inisiasi, menciptakan *random walk* pada posisi awal (*ZERO*), didalam border yang ditentukan dalam kode *world*, dan fungsi menyimpan *step_history*. Pada fungsi *walk algoritma* akan menuruti fungsi jumlah langkah dalam kode *world*, dengan kondisi: jika langkah lebih dari/sama dengan 4 atau dalam kemungkinan 25%, *algoritma* akan berganti arah langkah, jika belum memenuhi kondisi tersebut arah dalam *step_history* akan terus ditempuh, selain kedua kondisi tersebut (contoh: bertemu dengan tembok) maka mengganti arah langkah. Fungsi *step* hanya menggerakan perhitungan dari *steps_since_turn*. Fungsi *change_direction* mengambil salah satu arah dari variabel *DIRECTIONS* (kanan, atas, kiri, bawah) untuk digunakan saat kondisi sebelumnya terpenuhi.

```
extends Node
class_name RandomWalk

const DIRECTIONS = [Vector2.RIGHT, Vector2.UP, Vector2.LEFT, Vector2.DOWN]

var position = Vector2.ZERO
var direction = Vector2.RIGHT
var borders = Rect2()
var step_history = []
var steps_since_turn = 0

func _init(starting_position, new_borders):
    assert(new_borders.has_point(starting_position))
    position = starting_position
    step_history.append(position)
    borders = new_borders

func walk(steps):
    for step in steps:
        if randf() <= 0.25 or steps_since_turn >= 4:
            change_direction()

        if step():
            step_history.append(position)
        else:
            change_direction()
    return step_history
}
func step():
    var target_position = position + direction
    if borders.has_point(target_position):
        steps_since_turn += 1
        position = target_position
        return true
    else:
        return false

func change_direction():
    steps_since_turn = 0
    var directions = DIRECTIONS.duplicate()
    directions.erase(direction)
    directions.shuffle()
    direction = directions.pop_front()
    while not borders.has_point(position + direction):
        direction = directions.pop_front()
```

Gambar 16. Contoh kode program *Random walk*

Gambar 17 merupakan kode *world*, dibuatlah fungsi *generate_level* yang terhubung dengan kode *random walk*, dimana variabel *map* memerintahkan *algoritma* untuk mengambil langkah sebanyak 500, dan posisi dalam *map* diberi kode untuk mengaktifkan fungsi *autotile* pada *game* engine dengan *tileMap.set_cell (location, -1)* yang menentukan bahwa hasil dari langkah *algoritma* akan menghapus *tile* tembok. *Algoritma* juga akan melangkah dengan ukuran *border* setengah dari ukuran level agar lebih hasil kondens, terlihat pada variabel *walker*.



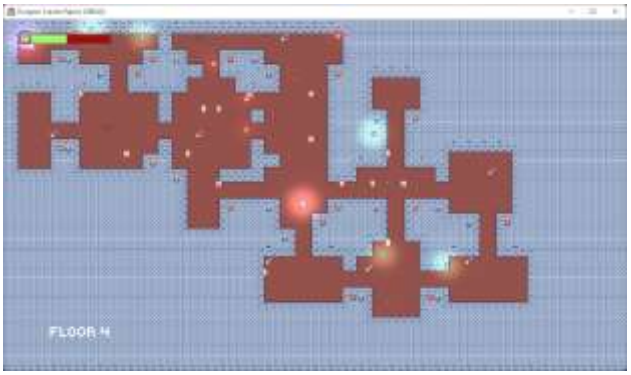
Gambar 17. Contoh hasil *Random walk* dengan 500 langkah

Hasil level terkesan terlalu berantakan, dengan beberapa ubin tunggal lepas, penulis ingin hasil level acak namun lebih rapi dan teratur. Untuk mengatasi ini, kode *random walk* dapat diubah dengan memberi variabel tambahan. Langkah pertama perubahan ini adalah mengganti kondisi "25% atau 4 langkah lalu berbelok" menjadi hanya "berbelok setelah 6 langkah".

Langkah berikutnya adalah variabel tambahan, variabel tersebut adalah *rooms* dengan tujuan membuat dan meletakan ruangan dengan ukuran yang ditentukan dalam fungsi *create_room* dan *place_room*. Sebagai tambahan, terdapat fungsi *get_end_room* yang akan digunakan untuk melacak ruangan terakhir yang dibuat, untuk meletakan *portal* pada *world* agar *player* dapat lanjut ke level yang dihasilkan selanjutnya.

Kemudian pada Gambar 18 menunjukan kode *world*, melakukan penyesuaian dengan mengurangi jumlah langkah *algoritma* menjadi 200 langkah. Posisi *player* akan berada pada langkah pertama dari *algoritma* dengan menggunakan kode *player.position = map.front* sementara *enemy* dan objek lain menggunakan *map[random.index]* untuk posisi acak serta tambahan variabel *range*(5) sehingga muncul 5 buah setiap level, dan *portal (exit)* menggunakan *walker.get_end_room* agar muncul diruangan terakhir yang dihasilkan *random walk*.

Hasil terlihat lebih teratur dengan jumlah ubin tunggal yang lebih sedikit, tambahan terakhir untuk level adalah efek kegelapan agar menambah kesan misteri.



Gambar 18. Contoh hasil *Random walk* 200 langkah dengan variabel rooms

3.4 *Testing* (Pengujian)

Tujuan utama dari pengujian ini adalah untuk memastikan bahwa semua elemen dan komponen dalam *game* berfungsi sesuai dengan yang diharapkan. Pengujian dilakukan untuk mengidentifikasi kelemahan dan kesalahan yang mungkin terjadi. Pengujian dibagi menjadi dua tahap utama: Pengujian *Alpha* dan Pengujian *Beta*.

a. Pengujian *Alpha*

Pengujian *Alpha* melibatkan pemeriksaan fungsionalitas dan kegunaan elemen *game*, termasuk UI dan animasi. Pengujian ini memastikan bahwa setiap fitur dan elemen dalam *game* bekerja sesuai dengan spesifikasi yang diharapkan. Berikut adalah beberapa poin penting dari hasil pengujian *Alpha*.

1) *Input* Kontrol Karakter: Validasi gerakan dasar dan interaksi menggunakan tombol yang ditentukan.
2) *Health Bar Player*: Validasi ukuran dan pengurangan *health bar* sesuai dengan serangan musuh.
3) Animasi *Player* dan Objek: Memastikan animasi berjalan lancar tanpa konflik.
4) *Tileset* dan UI Desain: Memastikan konsistensi dan keterhubungan *tileset* serta kejelasan navigasi UI.

b. Pengujian *Beta*

Pengujian *Beta* menggunakan metode *Blackbox* yang berfokus pada *input* dan *output* sistem tanpa memperhatikan bagaimana sistem mencapai hasil tersebut. Pengujian ini dibagi menjadi dua bagian: *User Interface* (UI) dan *Gameplay*.

1) Pengujian *User Interface:* Memastikan semua tombol pada menu utama, *pause menu*, dan *scene* lain bekerja dengan baik.
2) Pengujian *Gameplay*: Validasi interaksi objek dalam *game* seperti *Green Ghost*, *enemy*, *player*, *chest*, *barrel*, dan *portal*, serta memastikan *algorithm* untuk muat *level* secara acak berfungsi dengan baik.

Hasil dari pengujian *Beta* menunjukkan bahwa semua fitur dan elemen berfungsi dengan *valid*, memastikan *game* siap untuk tahap perilisan setelah pengujian *internal* ini.

3.5 *Release* (Perilisan)

Setelah *game* selesai dikembangkan dan diuji, langkah selanjutnya adalah merilis *game* tersebut. Gambar 19 menunjukkan hasil perilisan dilakukan dengan mengunggah *game* ke *website itch.io*

Gambar 19. Hasil unggah *game* pada *Website Itch.io*.

4. Kesimpulan

Penelitian ini berhasil menunjukkan implementasi Algoritma *Random Walk* dalam *Procedural Generation* untuk pembuatan *level* otomatis dan dinamis pada genre *game Dungeon Crawler* 2D. Berdasarkan pengujian menggunakan metode *Blackbox*, *algorithm Random Walk* mampu menghasilkan *level* yang selalu diacak setiap kali permainan dijalankan dan saat pemain memasuki *portal* ke *level* berikutnya. Hal ini menunjukkan bahwa Algoritma *Random Walk* efektif dalam menciptakan variasi *level* yang tinggi, sehingga meningkatkan *replayability* dan memberikan pengalaman bermain yang berbeda bagi pemain setiap kali memainkan *game*.

Meskipun demikian, terdapat beberapa area yang dapat ditingkatkan dalam penelitian selanjutnya. Pertama, optimasi algoritma diperlukan untuk meningkatkan efisiensi dan mengurangi waktu pemrosesan. Meskipun algoritma *Random Walk* efektif dalam menghasilkan variasi *level*, penelitian lebih lanjut diperlukan untuk mengoptimalkan performa algoritma. Kedua, implementasi *Random Walk* saat ini menghasilkan *level* secara acak tanpa mempertimbangkan tingkat kesulitan. Penelitian selanjutnya dapat fokus pada pengembangan metode untuk mengatur kesulitan *level* berdasarkan progresi pemain, sehingga tantangan yang diberikan dapat disesuaikan dengan kemampuan pemain. Ketiga, menambahkan elemen naratif atau tujuan spesifik dalam *level* yang dihasilkan dapat meningkatkan keterlibatan pemain. Penelitian masa depan bisa mengeksplorasi bagaimana mengintegrasikan cerita atau misi yang dinamis dalam *level* yang dihasilkan secara prosedural. Keempat, diversifikasi elemen lingkungan dan jenis musuh yang muncul dalam setiap *level* dapat lebih memperkaya pengalaman bermain. Penggunaan algoritma tambahan atau kombinasi dengan teknik lain bisa dieksplorasi untuk mencapai tujuan ini. Terakhir, mengadakan studi lebih mendalam mengenai pengalaman pengguna (*user experience*) terhadap *level* yang dihasilkan dapat memberikan wawasan lebih lanjut tentang aspek apa saja yang perlu diperbaiki atau ditambahkan. Ini termasuk *feedback* langsung dari pemain untuk penyesuaian lebih lanjut. Dengan mempertimbangkan area pengembangan di atas, penelitian ini dapat memberikan kontribusi yang lebih komprehensif dalam bidang *Procedural Generation* untuk *game*, khususnya dalam *genre Dungeon Crawler* 2D.

Daftar Pustaka

[1] M. Nguyen, "Fundamentals of 2D Game Art," 2021.

[2] J. Freiknecht, "Procedural content generation for games," 2021.

[3] N. Shaker, J. Togelius, and M. J. Nelson, "Procedural content generation in games," 2016.

[4] S. Riddle and O. C. Zecha, "Perlin Noise and Returning Results from Shader Programs," *Direct3D ShaderX*, p. 232.

[5] B. Barufaldi *et al.*, "Computational breast anatomy simulation using multi-scale Perlin noise," *IEEE Trans. Med. Imaging*, vol. 40, no. 12, pp. 3436–3445, 2021.

[6] W. R. Aisyiyah, "Simulasi 2 dimensi gambar berkabut berdasarkan Transmission Map Pada Dark Channel Pior (DCP) menggunakan citra Kawah Gunung Kelud," Universitas Islam Negeri Maulana Malik Ibrahim, 2019.

[7] D. A. Ramadhan and A. D. Indriyanti, "Procedural Content Generation pada Game World Exploration Sandbox Menggunakan Alogoritma Perlin Noise," *J. Informatics Comput. Sci.*, vol. 4, no. 01, pp. 86–91, 2022.

[8] C. Gasch, M. Chover, I. Remolar, and C. Rebollo, "Procedural modelling of terrains with constraints," *Multimed. Tools Appl.*, vol. 79, pp. 31125–31146, 2020.

[9] F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, and X. Kong, "Random walks: A review of algorithms and applications," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 2, pp. 95–107, 2019.

[10] X. Mao, K. Yuan, Y. Hu, Y. Gu, A. H. Sayed, and W. Yin, "Walkman: A communication-efficient random-walk algorithm for decentralized optimization," *IEEE Trans. Signal Process.*, vol. 68, pp. 2513–2528, 2020.

[11] M. Ijai and S. H. Suryawan, "Jurnal Computer Science and Information Technology (CoSciTech) Penerapan Algoritma Random Walk Untuk Procedural Map Pada Game ' The Last Hope ' 2D Application of the Random Walk Algorithm for Procedural Maps in the 2D Game ' The Last Hope ,'" vol. 4, no. 3, 2024.

[12] Shubham Pandey and R. K¨uhn, "A Random Walk Perspective on Hide-and-Seek Games," *J. Phys. A Math. Theor.*, pp. 0–23, 2019.

[13] R. Febriani and S. Suprijadi, "Aplikasi Metoda Random Walks untuk Kontrol Gerak Robot Berbasis Citra," *J. Otomasi Kontrol dan Instrumentasi*, vol. 2, no. 1, p. 21, 2011, doi: 10.5614/joki.2010.2.1.3.

[14] J. Sharp *et al.*, "Labyrinthos: A Dungeon Maze Game," 2019.

[15] A. Y. A. Aziz, "Implementasi Algoritme Procedural Content Generation Menggunakan Lindenmayer System untuk Membuat Dunia Tiga Dimensi bagi Permainan Bergenre Dungeon Crawler," Institut Teknologi Sepuluh Nopember, 2021.

[16] R. A. Krisdiawan and others, "Penerapan Model Pengembangan Gamegdlc (Game Development Life Cycle) Dalam Membangun Game Platform Berbasis Mobile," *Teknokom*, vol. 2, no. 1, pp. 31–40, 2019.