Available at https://ejournal3.undip.ac.id/index.php/jtk
DOI: 10.14710/jtk.v1i2.36673

Jurnal Teknik Komputer, Vol. 1, No. 2, 62-72

Implementasi *Procedural Content Generation* pada *Game* Menggunakan Unity

Implementation of Procedural Content Generation on Games Using Unity

Muhammad Hanif Atthariq, Dania Eridani, Adnan Fauzi

Program Studi Teknik Komputer, Fakultas Teknik, Universitas Diponegoro
Jl. Prof. Soedarto, SH, Kampus Undip Tembalang, Semarang, Indonesia 50275

How to cite: M. H. Atthariq, D. Eridani, A. Fauzi, "Implementasi *Procedural Content Generation* pada *Game* Menggunakan Unity", *Jurnal Teknik Komputer*, vol. 1, no. 2, pp. 62-72, November 2022, doi: 10.14710/jtk.v1i2.36673 [Online].

Abstract* – *Since the invention of video games, the concept of conventional games has gradually changed to encompass digital games, The term "games" is used to refer to video games. Procedural Content Generation (PCG) is the use of an algorithm to generate game content that would typically be produced by a human. Rogue, a game from the early 1980s, is frequently mentioned as the first game to feature a randomized map, hence the genre of games including randomized map is referred as "rogue-like".

The amount of time needed to create content for a game is a problem in game development, and the game development industry is interested in PCG as solution to provide endlessly new content. It is used as a method of level generation for rogue-like style levels. A level generator strives to produce straightforward, always-playable level designs with adequate variance.

Keywords *– Game; Procedural Content Generation; Game Development*

Abstrak* – *Sejak penemuan game digital, konsep permainan konvensional secara bertahap berubah menjadi permainan multimedia digital, Istilah "game" digunakan untuk merujuk pada permainan digital. Procedural Content Generation (PCG) adalah penggunaan algoritma untuk menghasilkan konten game yang biasanya diproduksi oleh manusia. Rogue, game digital dari awal 1980-an, disebut sebagai game pertama dengan generasi peta acak, maka genre game dengan peta acak disebut sebagai "rogue-like".

Jumlah waktu yang dibutuhkan untuk membuat konten untuk game merupakan masalah dalam pengembangan game, dan industri pengembangan game tertarik pada PCG sebagai solusinya untuk menyediakan konten baru tanpa henti. PCG digunakan sebagai metode untuk menghasilkan level untuk game bergenre rogue-like. Generator level berusaha untuk menghasilkan desain level yang sederhana dengan varian yang banyak.

Kata kunci *– Game; Procedural Content Generation; Pengembangan Game*

[1] Penulis Korespondensi (M. H. Atthariq)
Email: muhammadha@student.ce.undip.ac.id

I. PENDAHULUAN

Game merupakan salah satu media digital yang cukup populer, salah satu genre *game* mulai menjadi populer adalah genre *Rogue-like* dengan *game* seperti *The Binding of Isaac* dan *Hades*. Berasal dari *game* digital tahun 1980, *Rogue* dikenal sebagai salah satu *game* pertama yang mengimplementasikan fitur peta yang di generasi secara acak, itulah mengapa *game* yang memiliki fitur generasi peta secara acak disebut sebagai *game* bergenre *Rogue-like*.

Latar belakang penelitian ini adalah untuk meneliti penerapan dari metode *Procedural Content Generation* pada *game Rogue-like* yang dibuat menggunakan Unity. Pada *game* yang dibuat, *Procedural Content Generation* diimplementasikan untuk membantu dalam pembuatan konten sehingga dapat memotong waktu yang dibutuhkan untuk membuat konten *game*. Unity digunakan karena Unity merupakan aplikasi untuk pengembangan *game* yang tersedia dengan gratis.

Procedural Content Generation mengacu kepada pembuatan konten *game* menggunakan algoritma dengan kontribusi manusia yang terbatas atau tanpa kontribusi manusia. "Konten *game*" umumnya dipahami di sini dan mencakup level, peta, misi, tekstur, karakter, tanaman, dan struktur, tetapi bukan *game engine* sendiri atau perilaku *non-playable character* [1]. Metode untuk menghasilkan konten tersebut berasal dari sistem berbasis aturan. Motivasi yang sering dikutip untuk menggunakan PCG adalah meningkatkan *replayability game* atau memperkenalkan konten adaptif [2].

Berdasarkan latar belakang masalah tersebut, maka rumusan masalah dalam penelitian ini didasarkan pada; diperlukannya cara yang efisien untuk membuat peta level dalam jumlah besar, peta *game* perlu digenerasi secara acak sehingga pemain tidak dapat menghafal peta setiap level, dan diperlukan sistem untuk generasi peta secara acak pada *game*. Tujuan yang ingin dicapai dari Tugas Akhir ini adalah membuat algoritma *map generation* yang dapat membuat peta secara acak menggunakan metode PCG, dan mengetahui apa kelebihan dari PCG pada algoritma *map generation*.

Submitted: 18 November; Accepted: 22 November; Published: 30 November

Jurnal Teknik Komputer, Vol. 1, No. 2, 62-72

II. KAJIAN LITERATUR

A. Kajian Penelitian Terdahulu

Kajian penelitian terdahulu adalah penelusuran terhadap penelitian yang berhubungan dengan permasalahan yang sudah dibahas sebelumnya dan dapat dijadikan sebagai bahan kajian penelitian dengan permasalahan yang menyerupai dengan penelitian yang akan dilakukan sehingga dapat dijadikan referensi.

Berdasarkan penelitian yang sudah dilakukan pada 4 penelitian terdahulu, perbedaan yang didapat adalah metode penelitian pada *Procedural Content Generation* dan keluaran dari penelitian yang sudah dilakukan. Semua penelitian di atas berfokus pada aspek yang berbeda dari *Procedural Content Generation*, Julian Togelius bertujuan untuk menemukan apa yang dapat dicapai PCG dan dia menyimpulkan bahwa ada 3 tujuan utama PCG dan tantangan-tantangan yang harus dihadapi untuk mewujudkan tujuan tersebut [1]. *An Analog History of Procedural Content Generation* oleh Gillian Smith mengeksplorasi asal PCG dengan cara melihat sejarah analognya dan apa yang dapat dipelajari darinya, seperti mengendalikan keacakan dengan cara merancang dengan teliti setiap bagian konten yang akan dirakit oleh algoritma [2]. Mathias Paul Babin mengusulkan bahwa solusi untuk merancang *procedural content generator* adalah bahwa generator harus memiliki lima metrik kualitas utama yaitu *speed*, *reliability*, *controllability*, *expressivity*, dan *creativity* [3]. *Procedural Dungeon Generation: A Survey* mengklasifikasi beberapa makalah bertopik PCG dan menyimpulkan bahwa beberapa makalah sulit untuk dievaluasi karena kurangnya informasi, untuk menghindari masalah ini mereka menyarankan agar makalah yang diterbitkan harus mendefinisikan genre *game* atau *game* yang levelnya akan dihasilkan dan menentukan struktur level *game* [4], Berdasarkan 4 penelitian tersebut dapat dikumpulkan apa yang dibutuhkan untuk membuat algoritma *Procedural Content Generation*.

B. *Procedural Content Generation*

Julian Togelius mendefinisikan *Procedural Content Generation* (PCG) sebagai perangkat lunak komputer yang mampu "membuat konten *game* sendiri atau bersama-sama dengan satu atau lebih desainer manusia" [1]. PCG memiliki beberapa keunggulan, seperti mengurangi biaya produksi *game* dengan mengurangi kebutuhan desainer untuk menghasilkan konten, mengendalikan kesulitan permainan, dan meningkatkan *replayability game* [4].

Pertama-tama perlu dipahami properti generator konten prosedural dan bagaimana properti spesifik itu memanifestasikan dirinya dalam definisi masalah PCG. Bayangkan sebuah skenario di mana konten perlu dibuat selama permainan berjalan, masalah ini pada dasarnya membutuhkan generator untuk memiliki properti kecepatan generasi yang cepat. Ini menjelaskan berbagai properti yang diinginkan dalam sebuah generator [3]. Properti termasuk:

- *Speed*, seberapa cepat generator dapat menghasilkan kontennya.
- *Reliability*, seberapa konsisten generator dapat menghasilkan konten berkualitas tinggi.
- *Controllability*, menentukan seberapa banyak kontrol yang dialokasikan untuk algoritma eksternal atau perancang manusia atas konten yang dihasilkan.
- *Expressivity*, menggambarkan keragaman konten yang dihasilkan oleh generator.
- *Creativity/Believability*, seberapa baik konten generator menyerupai konten yang didesain oleh manusia.

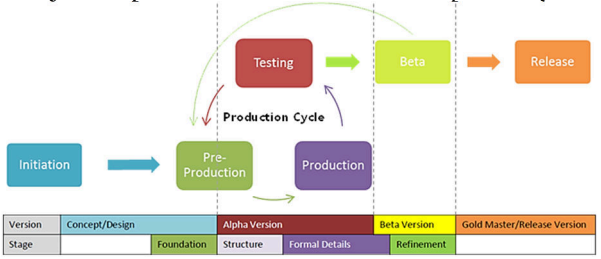
C. Unity

Salah satu pendiri Unity, David Helgason mendeskripsikan Unity sebagai *game engine* dan *Integrated Development Environment* (IDE) untuk membuat media interaktif seperti *video game*. Unity dapat digunakan untuk membuat *game* tiga dimensi (3D) dan dua dimensi (2D), serta simulasi interaktif dan media lainnya. Unity telah diadopsi oleh industri selain *game*, seperti film, otomotif, dan arsitektur. Unity dikenal dengan kemampuan *prototyping* yang cepat dengan target *publishing* yang luas [5].

Pada tahun 2005 versi pertama Unity dirilis oleh David Helgason, Joachim Ante, dan Nicholas Francis, dengan tujuan untuk "mendemokratisasikan" pengembangan *game* dengan membuatnya dapat diakses oleh lebih banyak developer, dengan membuat *game engine* yang mudah diakses, dengan set *tools* yang lengkap untuk pengembangan *game*, memiliki alur kerja yang sederhana, manajemen aset, dan antarmuka *drag-and-drop yang* sederhana.

III. METODOLOGI PENELITIAN

Metodologi penelitian untuk melakukan "Implementasi *Procedural Content Generation* pada *Game* Menggunakan Unity" menggunakan metodologi *Game Development Life Cycle* (GLDC), yang ditunjukkan pada Gambar 1. *Game Development Cycle* .



Gambar 1. *Game Development Cycle* [7]

Dibandingkan metode *Software Development Life Cycle* (SLDC), metode *Game Development Life Cycle* lebih terstruktur untuk pembuatan *game*, tahap-tahap dalam GLDC berbeda dibandingkan SLDC. Pada *Game Development Life Cycle* setelah tahap *Initiation* akan masuk ke siklus produksi atau *Production Cycle*, siklus

Submitted: 18 November; Accepted: 22 November; Published: 30 November

ini mencakupi 3 tahap selanjutnya yaitu tahap *Pre-Production*, *Production*, Testing. Setelah *Production Cycle* selesai maka *game* akan masuk ke tahap Beta, di mana *game* akan diuji oleh pihak ketiga, setelah hasil dari pengujian beta diimplementasikan dan *bug* diperbaiki, *game* akan dianggap siap untuk dirilis.

A. Inisiasi *(Initiation*)

Tahap pertama ini berupa pembuatan konsep awal *game*, penentuan genre *game*, dan penentuan target *audience* untuk *game*. Hasil dari tahap ini adalah konsep dasar dan deskripsi sederhana tentang *game* [6]. Konsep awal dari *game* akan menentukan bagaimana *game* dimainkan, dalam proses ini juga akan ditentukan *game engine* yang akan digunakan.

B. Pra-produksi (*Pre-production*)

Tahap ini merupakan pembuatan prototipe *game*, prototipe dibuat berdasarkan genre yang ditentukan dalam Tahap Inisiasi. Prototipe yang dibuat berfokus pada pembuatan mekanik utama *game*, mekanisme karakter, tingkat kesulitan *game*, dan dokumentasi dari *game* tersebut, prototipe *game* akan dibuat menggunakan aset-aset sederhana. Tahap ini berakhir ketika prototipe *game* telah disetujui [6].

C. Produksi (*Production*)

Pada tahap ini prototipe *game* yang telah dibuat pada Tahap Pra-produksi dikembangkan menjadi sebuah *game*. Tahap ini berkaitan dengan peningkatan fungsionalitas prototipe, pembuatan aset *game*, penambahan fitur baru, peningkatan kinerja secara keseluruhan, dan perbaikan *bug* yang tersisa dalam *game*.

D. Pengujian (*Testing*)

Tahap ini merupakan pengujian internal yang dilakukan untuk menguji keseluruhan *game*. Jika *bug* atau kegagalan ditemukan selama proses pengujian, skenario dianalisis untuk menentukan penyebab *bug*. Selain mencari *bug*, kualitas aksesibilitas *game* juga diuji, jika penguji menemukan *game* sulit untuk dimainkan atau dipahami, artinya aksesibilitas *game* tidak mencukupi. Hasil dari tahap pengujian menentukan apakah *game* dapat dilanjutkan ke tahap berikutnya atau kembali mengulangi siklus produksi [6].

E. Beta

Tahap ini akan dilakukan pengujian oleh pihak ketiga. Pengujian beta terus menggunakan metodologi pengujian yang sama seperti tahap sebelumnya. Selama tahap ini, penguji bebas memainkan *game*, mendokumentasikan pengalaman selama bermain dan *bug* yang ditemukan selama proses pengujian, dan mengembalikan hasil dokumentasi kepada developer untuk pembenaran *bug* dan penyempurnaan lebih lanjut. Hasil Tahap Beta berupa laporan *bug* dan masukan dari penguji, jika *bug* masih ditemukan selama Tahap Beta, siklus produksi akan diulang untuk memperbaiki *bug*.

F. Rilis (*Release*)

Pada tahap ini, *game* telah mencapai tahap akhir dan siap untuk dirilis. Tahap ini mencakup dokumentasi proyek, rencana pemeliharaan *game*, dan ekspansi *game* [6]. Sebuah *game* mencapai tahap ini hanya ketika masukan dari pengujian beta diimplementasikan dan *bug* diperbaiki sampai titik di mana tidak ada masalah lain yang ditemukan dalam Tahap Pengujian dan Beta.

IV. Hasil dan Pembahasan

Proses pengembangan *game* akan mengikuti *game development life cycle* yang memiliki 6 tahap, tetapi pada penelitian ini hanya digunakan 4 tahap pertama. Alasan mengapa hanya digunakan 4 tahap pertama adalah *game* yang dibuat pada penelitian ini tidak bertujuan untuk di rilis ke publik.

A. *Game Development Life Cycle*

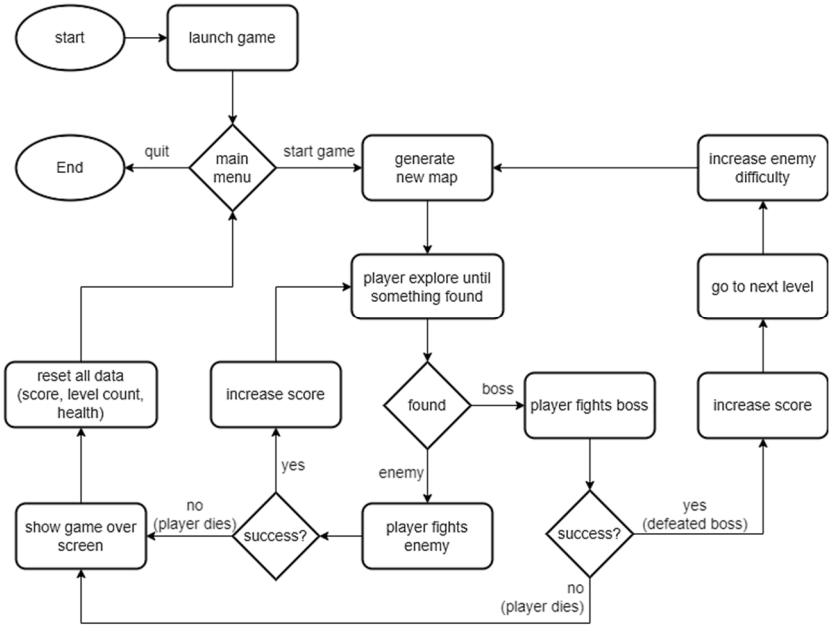
Dalam implementasi *procedural content generation* untuk *game*, akan digunakan 4 tahap pertama dari *game development life cycle* yang merupakan tahap inisiasi yang terutama berfokus pada pembuatan konsep awal *game*, tahap pra-produksi yang membuat prototipe untuk *game*, tahap produksi yaitu pembuatan *game* itu sendiri, dan tahap pengujian yaitu pengujian internal oleh developer untuk melihat apakah ada *bug* atau fitur yang kurang dalam *game*.

Inisiasi (Initiation)

Pertama konsep dasar dari *game* dibuat pada tahap Inisiasi, konsep yang didapatkan adalah *game* dua dimensi bergenre *Rogue-like* dengan tujuan untuk menyelesaikan level sebanyak mungkin dan mendapatkan skor tinggi, dengan konsep ini *game* berfokus untuk dapat memainkan beberapa level dalam waktu singkat, yang berarti diperlukan cara yang efisien untuk membuat jumlah peta level yang besar, untuk meraih ini, algoritma *procedural content generation* digunakan untuk menangani pembuatan peta. Pada tahap ini juga dibuatlah *flowchart* untuk mendeskripsikan alur *game* yang dapat dilihat pada Gambar 2. *Flowchart game*.

Game bergenre *Rogue-like* dikenal oleh peta acaknya yang sederhana dan dapat dimainkan berkali-kali, maka setiap level dibuat dengan ukuran kecil sehingga pemain dapat mengalami beberapa level dalam satu sesi permainan, ini dilakukan dengan cara membatasi ukuran peta untuk hanya memiliki 16 ruangan dalam kotak 4x4. Tujuan dari setiap level adalah untuk mencapai akhir level dan mengalahkan bos dan maju ke level selanjutnya, kesulitan *game* akan meningkat setiap kali pemain naik ke level berikutnya dengan cara *enemy health* dan *enemy damage*.

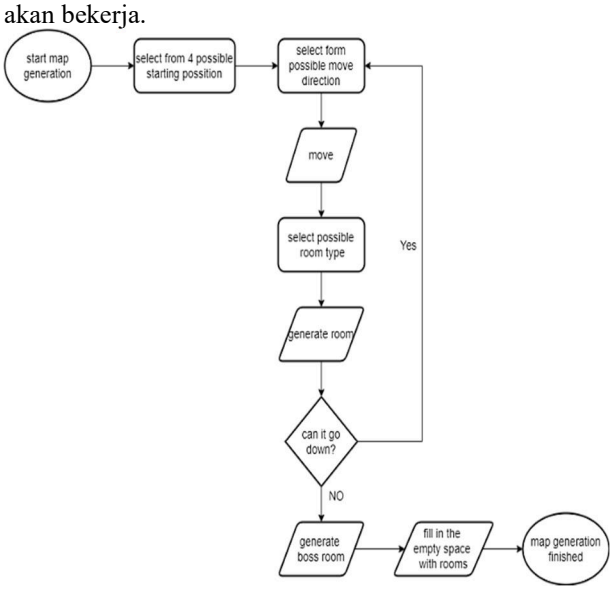
Submitted: 18 November; Accepted: 22 November; Published: 30 November



Gambar 2. *Flowchart game*

Pra-produksi (Pre-production)

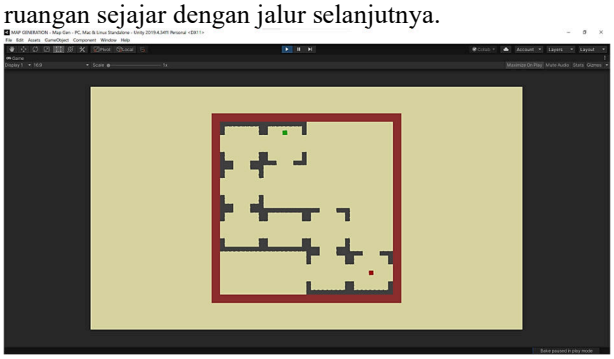
Tahap ini dimulai dengan mengimplementasikan konsep dasar fitur *map generation*, generasi level pada *game* menggunakan sebuah tipe *procedural content generation*, Gambar 3. *Flowchart* algoritma *Map Generation* menampilkan *flowchart* algoritma *Map Generation* yang menjelaskan bagaimana algoritma akan bekerja.



Gambar 3. *Flowchart* algoritma *Map Generation*

Aspek level yang dihasilkan secara prosedural adalah jalur peta dan ruangan yang akan ditemui pemain saat bermain *game*, menggunakan *procedural content generation* untuk membuat setiap level akan memastikan bahwa setiap permainan akan berbeda dan unik. Peta dibuat pada *grid* berukuran 40x40, dan setiap ruangan berukuran 10x10 sehingga setiap level memiliki total 16 ruangan. Pertama algoritma akan membuat jalur utama untuk memastikan bahwa selalu ada jalur yang jelas ke ujung tanpa jalan buntu, setelah itu akan mengisi sisa ruang kosong dengan ruangan acak.

Pada Gambar 4. Algoritma *Map Generation* menggenerasi jalur utama peta dapat dilihat versi prototipe pertama yang hanya menghasilkan jalur utama di dalam batas yang sudah di atur, hal ini dilakukan dengan menetapkan batas untuk setiap sisi pergerakan generator peta sehingga tidak menghasilkan ruangan di luar area yang telah ditetapkan. Setelah itu, setiap ruangan akan ditambahkan jalur untuk memasuki ruangan sehingga pemain dapat melewati dari satu ruangan ke ruangan lain, dan memastikan setiap jalur ruangan sejajar dengan jalur selanjutnya.



Gambar 4. Algoritma *Map Generation* menggenerasi jalur utama peta

Proses *Level generation* dimulai oleh LevelGeneration.cs pada `private void start` yang akan memilih posisi awal dengan cara memilih secara acak dari titik-titik yang sudah di atur pada `inspector` Unity menggunakan `int randStartingPos = Random.Range(0, startingPositions.Length)`, posisi titik yang terpilih didapatkan menggunakan `transform.position = startingPositions [randStartingPos].position` dan pada posisi itu akan dibuatlah *starting room* oleh `Instantiate(rooms[4],`

Submitted: 18 November; Accepted: 22 November; Published: 30 November

`transform.position, Quaternion. identity)`. Setelah *starting room* dibuat, akan dipilih suatu nomor secara acak dari 1 hingga 5 oleh `direction = Random.Range(1, 5)` untuk menentukan arah membuat ruangan selanjutnya, 1 dan 2 merupakan arah kanan, 3 dan 4 arah kiri, dan 5 menunjukkan arah bawah. Alasan mengapa terdapat 2 nomor untuk kiri dan kanan, dan hanya 1 untuk bawah adalah sehingga lebih sering untuk *map* dibuat secara horizontal dibandingkan turun sehingga membuat *map* lebih panjang, semua ini dapat dilihat pada Gambar 5. Skrip LevelGeneration.cs bagian pemilihan posisi ruang awal.

```
private void Start()
    {
        pauseMenu.LoadingStart();

        int randStartingPos = Random.Range(0, startingPositions.
Length);
        transform.position = startingPositions[randStartingPos].
position;
        Instantiate(rooms[4], transform.position, Quaternion.
identity);

        direction = Random.Range(1, 5);
    }
```

Gambar 5. Skrip LevelGeneration.cs bagian pemilihan posisi ruang awal

Pada `private void Update`, akan dipanggil *function Move* dan *timer* pada setiap gerakan untuk memberikan waktu untuk membuat ruangan, *source code* untuk *timer* dapat dilihat pada Gambar 6. Skrip LevelGeneration.cs bagian *timer* di antara pembuatan ruangan, *timer* tersebut akan diatur oleh `timeBtwRoom` dan nilainya dapat diubah pada *inspector* Unity, karena `void Update` memperbarui setiap *frame*, ini memungkinkan untuk membuat *timer* menggunakan `timeBtwRoom -= Time.deltaTime` kode tersebut akan mengurangi nilai `timeBtwRoom` hingga mencapai 0 dan akan melalukan reset menggunakan `timeBtwRoom = startTimeBtwRoom`.

```
private void Update()
    {
        if (timeBtwRoom <= 0 && stopGeneration == false)
        {
            Move();
            timeBtwRoom = startTimeBtwRoom;
        }
        else
        {
            timeBtwRoom -= Time.deltaTime;
        }
    }
```

Gambar 6. Skrip LevelGeneration.cs bagian *timer* di antara pembuatan ruangan

Function Move terdiri dari 3 *if statement* yang akan menentukan apa yang akan terjadi untuk setiap arah yang dipilih. Bagian pertama adalah untuk jika bergerak ke kanan, pertama akan memeriksa apakah posisi saat ini berada di tepi kanan peta menggunakan `if (transform.position.x < maxX)`, jika iya maka nilai `direction` akan diubah menjadi 5 untuk memindahkannya ke bawah bukan ke kanan, jika tidak maka akan mendapatkan posisi baru menggunakan `Vector2 newPos = new Vector2(transform.position.x + moveAmount, transform.position.y)` dan pindah ke posisi tersebut menggunakan `transform.position = newPos`, kemudian akan menghasilkan nomor acak antara 0 dan 3 untuk memilih jenis ruangan dan membuat ruangan yang dipilih menggunakan `Instantiate(rooms[rand], transform.position,Quaternion.identity)`.

Setelah ruangan dibuat, dia akan kembali memilih arah secara acak untuk bergerak, pernyataan *if* dibuat untuk mencegahnya bergerak ke kiri sehingga tidak akan kembali dan menghasilkan ruang di atas yang sebelumnya, semua ini dapat dilihat pada Gambar 7. Skrip LevelGeneration.cs bagian fungsi bergerak ke kanan.

```
private void Move()
    {
        if (direction == 1 || direction == 2)
        {  // Move RIGHT !
            if (transform.position.x < maxX)
            {
                Vector2 newPos = new Vector2(transform.position.x +
moveAmount, transform.position.y);
                transform.position = newPos;

                int rand = Random.Range(0, 3);
                Instantiate(rooms[rand], transform.position,
Quaternion.identity);

                direction = Random.Range(1, 5);
                if (direction == 3)
                {
                    direction = 2;
                }
                else if (direction == 4)
                {
                    direction = 5;
                }
            }
            else
            {
                direction = 5;
            }
        }
```

Gambar 7. Skrip LevelGeneration.cs bagian fungsi bergerak ke kanan

Gambar 8. Skrip LevelGeneration.cs bagian fungsi bergerak ke kiri menjelaskan jika algoritma bergerak ke kiri sangat mirip dengan bergerak ke kanan dengan sedikit perubahan seperti memeriksa `minX` dibandingkan `maxX` untuk melihat apakah posisi saat ini berada di tepi kiri peta, dan nomor arah acak hanya dari 3 hingga 5 dibandingkan 1 sampai 5, ini untuk mencegah bergerak ke kanan.

```
else if (direction == 3 || direction == 4)
        {  // Move LEFT !
            if (transform.position.x > minX)
            {
                downCounter = 0;
                Vector2 newPos = new Vector2(transform.position.x -
moveAmount, transform.position.y);
                transform.position = newPos;

                int rand = Random.Range(0, 3);
                Instantiate(rooms[rand], transform.position,
Quaternion.identity);

                direction = Random.Range(3, 5);
            }
            else
            {
                direction = 5;
            }
        }
```

Gambar 8. Skrip LevelGeneration.cs bagian fungsi bergerak ke kiri

Bagian terakhir adalah bergerak ke bawah, di sini terdapat variabel baru bernama `downCounter`, variabel ini menghitung berapa kali pada proses generasi peta dia telah bergerak ke bawah, jika telah turun dua kali berturut-turut dia akan memeriksa apakah ruangan

Submitted: 18 November; Accepted: 22 November; Published: 30 November

sebelumnya memiliki jalur ke bawah atau tidak, jika tidak ada maka akan diganti dengan tipe ruangan yang memiliki jalur ke bawah, *code* untuk bagian ini dapat dilihat pada Gambar 9. Skrip LevelGeneration.cs bagian fungsi bergerak ke bawah.

```
else if (direction == 5)
        {  // Move DOWN !
            downCounter++;

            if (transform.position.y > minY)
            {
                Collider2D roomDetection =
Physics2D.OverlapCircle(transform.position, 1, room);
                if (roomDetection.GetComponent<RoomType>().type != 1
&& roomDetection.GetComponent<RoomType>().type != 3)
                {
                    if (downCounter >= 2)
                    {

roomDetection.GetComponent<RoomType>().RoomDestruction();
                        Instantiate(rooms[3], transform.position,
Quaternion.identity);
                    }
                    else
                    {

roomDetection.GetComponent<RoomType>().RoomDestruction();

                        int randBottomRoom = Random.Range(1, 3);
                        if (randBottomRoom == 2)
                        {
                            randBottomRoom = 1;
                        }
                        Instantiate(rooms[randBottomRoom],
transform.position, Quaternion.identity);
                    }
                }

                Vector2 newPos = new Vector2(transform.position.x,
transform.position.y - moveAmount);
                transform.position = newPos;

                int rand = Random.Range(2, 3);
                Instantiate(rooms[rand], transform.position,
Quaternion.identity);

                direction = Random.Range(1, 5);
            }
```

Gambar 9. Skrip LevelGeneration.cs bagian fungsi bergerak ke bawah

Jika algoritma bergerak ke bawah melewati batas `minY` proses generasi peta akan berhenti dan mengganti ruangan terakhir dengan ruangan khusus yang disebut ruangan bos, *source code* tersebut dapat dilihat pada Gambar 10. Skrip LevelGeneration.cs bagian menghentikan generasi peta. Ketika jalur utama selesai, sisa *grid* akan diisi dengan ruangan acak menggunakan SpawnRoom.cs.

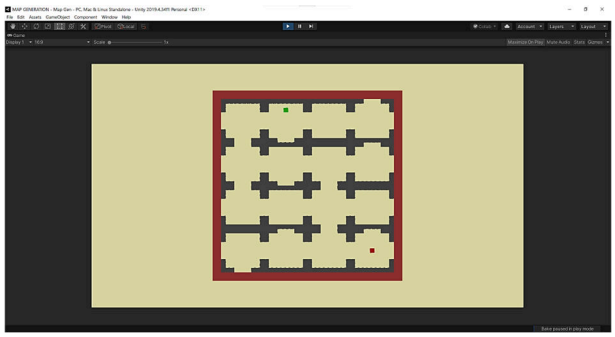
```
else
    {   // STOP LEVEL GENERATION !
        stopGeneration = true;
        pauseMenu.LoadingEnd();

        Collider2D roomDetection =
Physics2D.OverlapCircle(transform.position, 1, room);
        roomDetection.GetComponent<RoomType>().RoomDestruction();
        Instantiate(rooms[5], transform.position,
Quaternion.identity);
    }
```

Gambar 10. Skrip LevelGeneration.cs bagian menghentikan generasi peta

Pertama algoritma akan memutuskan apa yang dianggap sebagai Room menggunakan `whatIsRoom` yang akan mendeteksi *layer mask* objek apakah disetel ke layer Room atau tidak, setelah mengetahui apa itu Room, dia akan melihat di mana dalam *grid* ruangan tidak dihasilkan menggunakan `Collider2D roomDetection = Physics2D.OverlapCircle (transform. position, 1, whatIsRoom)`, setelah itu akan mulai membuat ruangan secara acak di ruang kosong. Semua proses generasi peta akan disembunyikan dari pemain oleh layar *loading* yang dimulai oleh `pauseMenu.LoadingStart()` di awal skrip, dan berakhir setelah generasi selesai menggunakan `pauseMenu.LoadingEnd()`, *source code* SpawnRoom.cs dapat dilihat pada Gambar 11. Skrip SpawnRoom.cs dan hasil algoritma *Map Generation* dilihat pada Gambar 12. Algoritma *Map Generation* mengisi ruang kosong.

```
    public LayerMask whatIsRoom;
    public LevelGeneration levelGen;

    void Update()
    {
        Collider2D roomDetection =
Physics2D.OverlapCircle(transform.position, 1, whatIsRoom);
        if (roomDetection == null && levelGen.stopGeneration == true)
        {// SPAWN RANDOM ROOM !
            int rand = Random.Range(0, 3);
            Instantiate(levelGen.rooms[rand], transform.position,
Quaternion.identity);
            Destroy(gameObject);
        }
    }
```

Gambar 11. Skrip SpawnRoom.cs



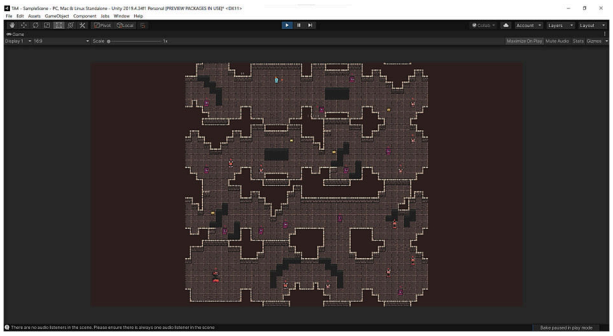
Gambar 12. Algoritma *Map Generation* mengisi ruang kosong

Fitur terakhir yang dibuat dalam prototipe adalah cara untuk maju ke level berikutnya, metode yang dihasilkan adalah dengan membuat musuh bos yang harus dilawan oleh pemain di akhir setiap level, dan ketika bos dikalahkan, pemain dapat naik ke level selanjutnya.

Produksi (Production)

Pada tahap Produksi prototipe akan dikembangkan menjadi *game* yang sebenarnya, dimulai dari menambahkan semua fitur dasar untuk *game* seperti sistem bergerak untuk pemain, sistem *combat*, sistem *health*, dan AI musuh, semua ini dibuat untuk memungkinkan pengujian dan pengembangkan *game* lebih lanjut. dalam tahap ini dimulai juga menambahkan *sprite* dan animasi untuk objek *game*, efek suara dan musik, untuk menambahkan variasi, musuh di dalam setiap ruang akan dihasilkan secara acak dari 3 jenis musuh yang ada. Fitur lain seperti antarmuka pengguna untuk menampilkan hal-hal seperti *health* pemain, penghitung skor pemain, tingkat level, dan penghitung waktu untuk setiap kemampuan pemain juga ditambahkan. Hasil pengembangan yang dilakukan pada tahap Produksi dapat dilihat pada Gambar 13. Hasil akhir algoritma *Map Generation*.

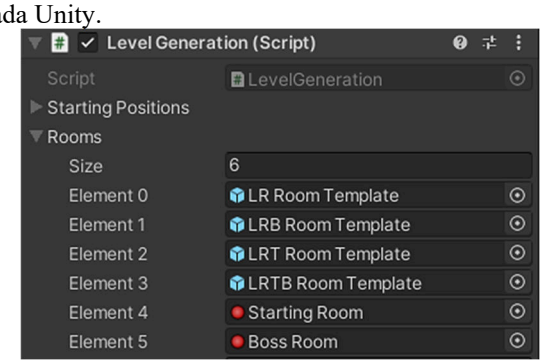
Submitted: 18 November; Accepted: 22 November; Published: 30 November



Gambar 13. Hasil akhir algoritma *Map Generation*

Melanjutkan pembuatan *procedural content generation* untuk peta, dibuatlah beberapa versi untuk setiap ruangan untuk menambahkan variasi, ada 12 versi dari setiap tipe ruangan sehingga ada total 48 ruangan di dalam *game*.

Ruangan yang dihasilkan ditentukan oleh tipe ruangan yang dipilih oleh skrip LevelGeneration.cs, ruangan diidentifikasi menggunakan angka, tipe ruangan 0 memiliki jalur ke kiri dan kanan, tipe 1 memiliki jalur ke kiri kanan dan bawah, tipe 2 memiliki jalur ke kiri kanan dan atas, dan tipe 3 memiliki jalur ke kiri kanan atas dan bawah. Ada juga 2 tipe ruangan khusus yaitu ruang awal dan ruang bos, semua ini dapat diatur menggunakan *Inspector* pada Unity yang dapat dilihat pada Gambar 14. *Inspector* LevelGeneration.cs pada Unity.



Gambar 14. *Inspector* LevelGeneration.cs pada Unity

Setiap tipe ruangan memiliki 12 varian untuk dipilih, dan ditentukan oleh skrip SpawnObject.cs, `int rand = Random.Range(0, objects.Length)` menghasilkan nomor secara acak dari 0 hingga 11, setelah itu akan membuat ruangan tersebut. Skrip SpawnObject.cs juga digunakan di bagian lain *game* seperti untuk menghasilkan jenis musuh yang muncul secara acak, skrip SpawnObject.cs dapat dilihat pada Gambar 15. Skrip SpawnObject.cs.

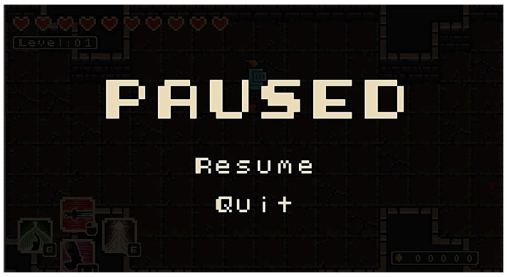
```
public GameObject[] objects;

    private void Start()
    {
        int rand = Random.Range(0, objects.Length);
        GameObject instance = (GameObject)Instantiate(objects[rand],
transform.position, Quaternion.identity);
        instance.transform.parent = transform;
    }
```

Gambar 15. Skrip SpawnObject.cs

Bagian akhir dari tahap produksi adalah membuat menu interaktif untuk permainan seperti menu utama, menu pause, layar mengalahkan level, layar kalah, dan *loading screen*. Gambar 16. Menu Utama menunjukkan Menu Utama atau Main Menu yang memiliki tiga tombol yaitu Start *Game*, *Options*, dan *Quit*.

Gambar 16. Menu Utama

Gambar 17. Menu Pause adalah Menu Pause yang akan muncul jika pemain menekan tombol *Escape* pada *keyboard*. Menu Pause memiliki 2 tombol yaitu Resume untuk kembali bermain dan *Quit* untuk kembali ke Menu Utama.



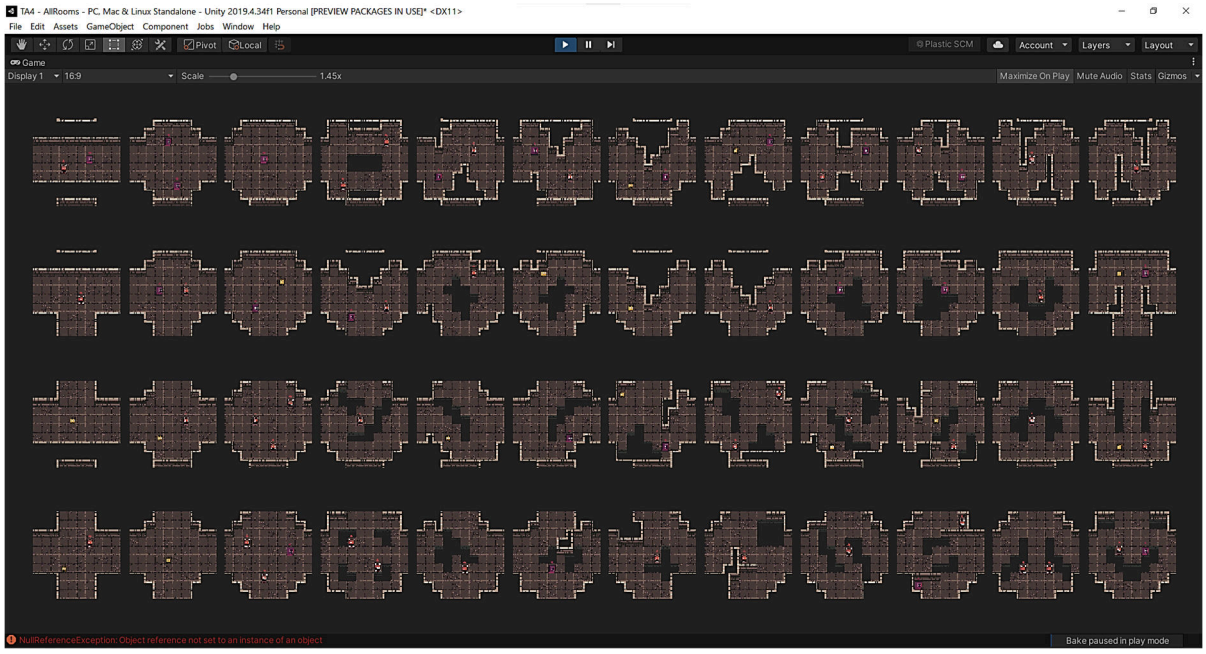
Gambar 17. Menu Pause

Saat mengalahkan level maka *game* akan menunjukkan layar yang memberi pilihan untuk maju ke level selanjutnya atau kembali ke Menu Utama, layar ini dapat dilihat pada Gambar 18. Layar mengalahkan level.

Gambar 18. Layar mengalahkan level

Jika pemain gagal untuk mengalahkan level maka *game* akan menunjukkan layar yang memberi pilihan untuk mencoba kembali atau ke Menu Utama, layar ini dapat dilihat pada Gambar 19. .

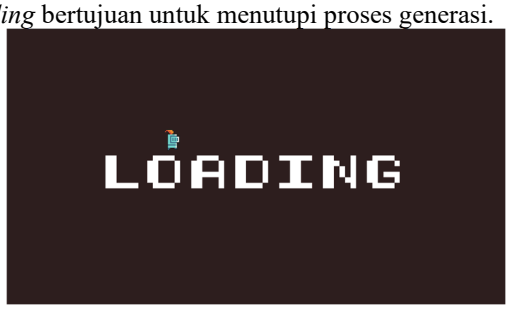
Submitted: 18 November; Accepted: 22 November; Published: 30 November



Gambar 21. Ruangan untuk digunakan oleh algoritma

Gambar 19. Layar kalah

Gambar 20. *Loading screen* menampilkan layar *loading* atau *Loading Screen* yang akan muncul saat *game* dalam proses menggenerasi peta level, layar *loading* bertujuan untuk menutupi proses generasi.



Gambar 20. *Loading screen*

Pengujian (Testing)

Pada tahap pengujian semua fitur yang telah dibuat diuji untuk memeriksa apakah ada *bug* atau fitur yang rusak, tetapi pengujian tidak hanya dilakukan di akhir *Game Development Life Cycle*, pengujian dimulai dari awal pembuatan *game* [8], tahap pengujian hanya menguji produk akhir yang akan di rilis. Metode yang digunakan dalam proses ini adalah *Functionality Testing* atau pengujian fungsionalitas. Semua proses pengujian dilakukan secara internal oleh developer.

Pengujian Fungsionalitas *game* berarti mengidentifikasi *bug*/cacat/kesalahan apa pun dalam *game* yang dapat memengaruhi pengalaman pengguna akhir, untuk menguji fungsionalitas algoritma *map generation*, algoritma diminta untuk menghasilkan 5 peta berturut-turut dan setiap peta dianalisis untuk menentukan apakah algoritma berfungsi sesuai yang diinginkan.

Pada Gambar 21. Ruangan untuk digunakan oleh algoritma dapat dilihat 48 ruangan berbeda, baris pertama menunjukkan tipe ruangan LR yang berarti ruangan tersebut memiliki hanya jalur ke kiri dan ke kanan. Kedua adalah tipe ruangan LRB yang memiliki jalur kiri, kanan, dan bawah. Baris ketiga adalah ruangan bertipe LRT yang memiliki jalur ke kiri, kanan, dan atas. Dan terakhir baris keempat adalah tipe LRTB yang berarti ruangan tersebut memiliki jalur ke semua arah.

Cara kerja algoritma adalah dengan mengambil ruangan dari sekumpulan ruangan yang telah dibuat seperti yang ditunjukkan pada Gambar 21. Ruangan untuk digunakan oleh algoritma dan menyusunnya dalam *grid* 4x4 untuk membentuk peta. Pertama, algoritma membuat jalur utama peta untuk memastikan selalu ada jalur ke ruang bos, kemudian mengisi bagian yang kosong dengan ruang acak. Hasil yang diinginkan dari setiap sesi pengujian adalah algoritma harus dapat menghasilkan peta unik yang berbeda dari satu sama lain, dan peta ini harus bisa diselesaikan tanpa jalan buntu.

Sebelum pengujian dilakukan ditemukan kesalahan pada algoritma, ditemukan bahwa algoritma lebih cenderung memilih jalur ke bawah dibandingkan ke kiri atau kanan, ini dikarenakan algoritma *map generation* bergerak berdasarkan nomor acak yang dipilih, nomor tersebut terkait dengan arah algoritma akan bergerak, nomor 1 dan 2 untuk bergerak ke kanan, 3 dan 4 untuk ke kiri, dan nomor 5 untuk ke bawah; algoritma *map*

Submitted: 18 November; Accepted: 22 November; Published: 30 November

generation seharusnya hanya bisa memilih di antara 5 nomor tersebut, tetapi ada kesalahan pada algoritma di mana dia dapat memilih dari 1 hingga 10 dan semua nomor di atas 5 dianggap sebagai bergerak ke bawah, inilah mengapa algoritma lebih cenderung bergerak ke bawah; kesalahan ini telah diperbaiki dengan cara menurunkan batas nomor yang dapat dipilih dari 10 menjadi 5.

Pada pengujian pertama algoritma menghasilkan peta dengan sempurna. Pertama, jalur utama peta dibuat tanpa jalan buntu ke ruang bos. Variasi ruangan yang dipilih oleh algoritma untuk jalur utama bagus, menggunakan 4 ruangan berbeda dari 48 opsi yang ada, hasil tahap pertama ini dapat dilihat pada Gambar 22. Jalur utama peta Pengujian 1.

Gambar 22. Jalur utama peta Pengujian 1

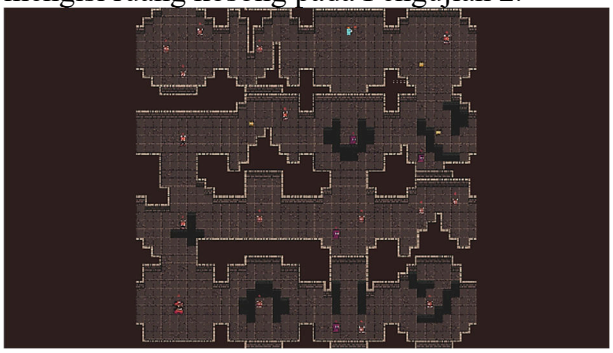
Algoritma kemudian mengisi ruang kosong yang tersisa dengan 10 ruangan. Variasi ruang yang dipilih untuk ini juga baik, menggunakan 9 ruangan berbeda dan hanya 1 ruangan berulang dari 48 opsi yang ada, hasil dari tahap ini dapat dilihat pada Gambar 23. Algoritma mengisi ruang kosong pada Pengujian 1.

Gambar 23. Algoritma mengisi ruang kosong pada Pengujian 1

Pada pengujian kedua algoritma menghasilkan jalur utama yang lebih panjang dari Pengujian 1, namun jalur yang dihasilkan tetap tidak memiliki jalan buntu menuju ruang bos. Variasi ruang yang dipilih untuk jalur utama bagus, menggunakan 6 ruangan yang berbeda, hasil tahap pertama ditunjukkan pada Gambar 24. Jalur utama peta Pengujian 2.

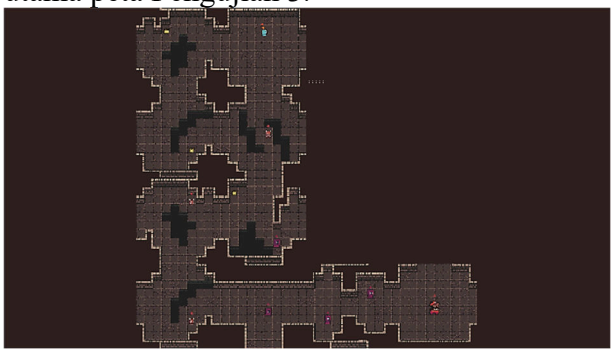
Gambar 24. Jalur utama peta Pengujian 2

Saat menghasilkan sisa ruangan, algoritma menghasilkan 8 ruangan untuk mengisi ruang kosong. Variasi ruang yang dipilih bagus, menggunakan 8 ruangan berbeda dan tidak ada yang sama, hasil dari tahap ini dapat dilihat pada Gambar 25. Algoritma mengisi ruang kosong pada Pengujian 2.



Gambar 25. Algoritma mengisi ruang kosong pada Pengujian 2

Peta ketiga dimulai dari titik yang sama seperti peta Pengujian 1, tetapi jalur yang dihasilkan tetap berbeda dari Pengujian 1. Pada pengujian ini variasi ruang yang dipilih untuk jalur utama tidak sebagus pengujian sebelumnya, menggunakan 6 ruangan yang berbeda dan 1 ruangan berulang, walaupun ada ruangan yang berulang dapat dilihat bahwa tipe musuh yang digenerasi di kedua ruangan tersebut tetap berbeda, hasil tahap pertama ini ditunjukkan pada Gambar 26. Jalur utama peta Pengujian 3.



Gambar 26. Jalur utama peta Pengujian 3

Algoritma menghasilkan 6 ruangan untuk mengisi ruang kosong. Dalam sisi variasi ruangan, peta pada pengujian ketiga ini hanya memiliki 9 ruangan unik dan

Submitted: 18 November; Accepted: 22 November; Published: 30 November

2 jenis ruangan yang berulang, hasil dari tahap ini dapat dilihat pada Gambar 27. Algoritma mengisi ruang kosong pada Pengujian 3.

Gambar 27. Algoritma mengisi ruang kosong pada Pengujian 3

Variasi ruang yang pada pengujian ini lebih buruk dibandingkan sebelumnya, terdapat 1 jenis ruangan yang berulang dan jenis ruangan tersebut juga berada pada jalur utama, tetapi seperti jenis ruangan yang berulang pada jalur utama tipe musuh yang berada pada ketiga ruangan tersebut memiliki kombinasi yang berbeda, ketiga ruangan dapat dilihat pada Gambar 28. .

Gambar 28. Tiga ruangan yang sama jenis

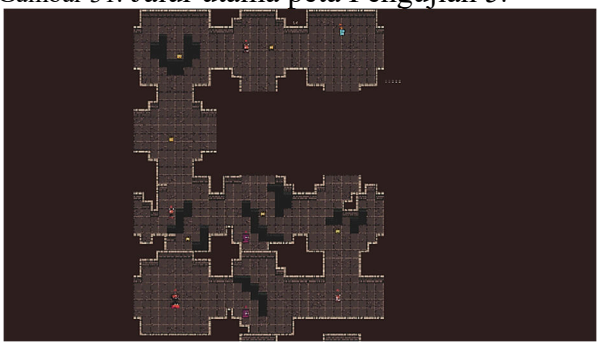
Pada pengujian keempat bekerja sesuai yang diinginkan, variasi ruangan yang dipilih untuk jalur utama bagus, menggunakan 7 ruangan yang berbeda, hasil tahap pertama ini ditunjukkan pada Gambar 29. Jalur utama peta Pengujian 4.

Gambar 29. Jalur utama peta Pengujian 4

Saat mengisi sisa ruangan, algoritma menghasilkan 7 ruangan untuk mengisi ruang kosong. Variasi ruang pada tahap ini masih memiliki 1 ruangan berulang, secara total algoritma menggunakan 6 ruangan berbeda dan 1 ruangan berulang pada tahap ini, hasil dari tahap ini dapat dilihat pada Gambar 30. Algoritma mengisi ruang kosong pada Pengujian 4.

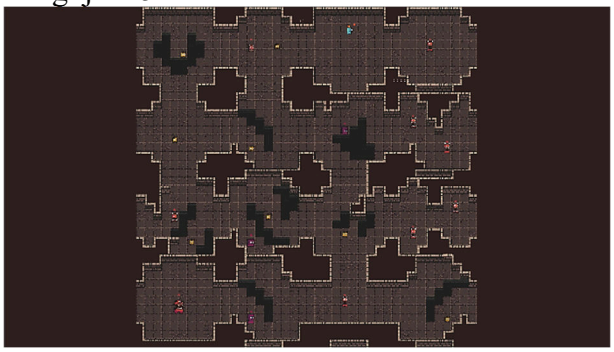
Gambar 30. Algoritma mengisi ruang kosong pada Pengujian 4

Pada pengujian terakhir, algoritma bekerja sesuai yang diinginkan, jalur utama yang digenerasi memiliki variasi ruangan yang bagus, memiliki 8 ruangan yang berbeda, hasil tahap pertama ini ditunjukkan pada Gambar 31. Jalur utama peta Pengujian 5.



Gambar 31. Jalur utama peta Pengujian 5

Saat mengisi sisa ruangan, algoritma menghasilkan 6 ruangan untuk mengisi ruang kosong. Suatu yang menarik terjadi pada pengujian ini, dalam proses pengisian ruang kosong terbuat jalur yang lebih pendek menuju ruang bos dibandingkan jalur utama yang dibuat pada tahap sebelumnya. Variasi ruang pada tahap ini memiliki 1 ruangan yang sama dengan salah satu ruangan dari jalur utama, secara total algoritma menggunakan 13 ruangan berbeda dan 1 ruangan berulang, hasil tahap ini dapat dilihat pada Gambar 32. Algoritma mengisi ruang kosong pada Pengujian 5.



Gambar 32. Algoritma mengisi ruang kosong pada Pengujian 5

Submitted: 18 November; Accepted: 22 November; Published: 30 November

Kesimpulan yang didapatkan dari kelima pengujian yang dilakukan adalah bahwa algoritma dapat menggenerasi peta secara konsisten tanpa jalan buntu ataupun *error*; jalur pada kelima pengujian semua berbeda dan kombinasi dari arah jalur utama jenis ruangan yang dipilih, algoritma *map generation* dapat membuat jumlah peta yang hampir tak terbatas. Variasi ruangan yang digenerasi algoritma walau masih terdapat jenis ruangan yang sama dalam satu peta sudah dapat dibilang bagus, dan masalah ini dapat di perbaiki dengan cara menambahkan jenis ruangan yang dapat digunakan oleh algoritma *map generation*.

B. *Procedural Content Generator*

Procedural content generator harus memenuhi lima metrik kualitas *speed*, *reliability*, *controllability*, *expressivity*, dan *creativity/believability*, menurut "*A Hybrid Approach to Procedural Dungeon Generation*" oleh Mathias Paul Babin [3]. Kelima metrik tersebut telah tercapai oleh algoritma *procedural content generation* yang dikembangkan pada penelitian ini.

Speed

Algoritma *procedural content generation* yang dihasilkan dapat memproduksi konten lebih cepat dan efisien daripada memproduksi konten secara manual, dibandingkan dengan menempatkan setiap ruangan pada setiap peta untuk sekian banyak level secara manual, algoritma dapat mengambil konten dasar yang telah disediakan, yaitu 48 ruangan dan 3 musuh yang telah dibuat oleh *designer* dan menghasilkan peta yang dibuat secara acak dengan menempatkan ruangan dan musuh secara acak dan memastikan bahwa level dapat diselesaikan. Algoritma *procedural content generation* yang telah dibuat dapat menghasilkan peta dalam beberapa detik.

Reliability

Peta yang dihasilkan oleh algoritma *procedural content generation* akan selalu memiliki jalur yang jelas hingga akhir, dan mencegah terbuatnya jalan buntu. Selama tahap pengujian, algoritma telah terbukti andal mampu menghasilkan peta tanpa kesalahan dan *error*.

Controllability

Designer dapat memodifikasi algoritma *procedural content generation* di dalam Unity, ini membantu meningkatkan skalabilitas *game* dengan cara memfasilitasi penambahan konten baru, yang perlu dilakukan *designer* hanyalah membuat ruangan dan musuh baru dan menambahkannya ke dalam algoritma; dari sana algoritma dapat menghasilkan peta menggunakan konten baru yang ditambahkan. Karena algoritma *procedural content generation* mengelola pembuatan peta, ukuran peta juga dapat ditingkatkan dengan mengubah parameter algoritma, dengan itu memudahkan untuk meningkatkan skala *game*.

Expressivity

Dalam penelitian ini, algoritma *procedural content generation* digunakan pada berbagai sistem seperti; menghasilkan jalur peta, menghasilkan ruangan secara acak, musuh untuk setiap ruangan, dan memilih *sprite* untuk ruangan. Dengan menggunakan algoritma *procedural content generation* sebagai dasar untuk beberapa sistem berbeda, proses pengembangan *game* menjadi lebih mudah dengan cara mengurangi waktu yang dibutuhkan untuk membuat sistem-sistem ini.

Creativity/Believability

Karena cara peta disusun, peta yang dihasilkan oleh algoritma *procedural content generation* akan terlihat identik dengan peta yang dibuat secara manual oleh manusia. Ini dikarenakan bagaimana *game* didesain, menggunakan *grid* 4x4 yang terdiri dari ruangan sebagai dasar untuk peta, ini memungkinkan algoritma *procedural content generation* untuk membangun peta yang tampak seperti dibuat oleh manusia dengan menempatkan ruangan-ruangan secara manual di *grid*.

V. KESIMPULAN

Berdasarkan hasil penelitian "Implementasi *Procedural Content Generation* pada *game* menggunakan Unity" yang telah dilakukan, didapatkan kesimpulan berikut; metode *Procedural Content Generation* berhasil diimplementasikan dalam *game* sebagai algoritma *map generation* pada *game*; kelebihan dari *Procedural Content Generation* termasuk membantu proses pengembangan *game* dengan membuat konten seperti peta dalam jumlah besar lebih cepat, dan mempermudah peningkatan skala *game*; algoritma *Procedural Content Generation* yang telah dibuat memenuhi lima metrik kualitas yang diajukan oleh Mathias Paul Babin. Algoritma dapat memproduksi hasil dengan cepat, dapat memproduksi hasil secara konsisten tanpa error, dapat dikontrol oleh designer berdasarkan konten dasar yang diberi, dapat menghasilkan beragam konten yang berbeda, konten yang dihasilkan oleh algoritma tidak jauh berbeda dari konten yang didesain oleh manusia.

UCAPAN TERIMA KASIH

Dengan selesainya pembuatan laporan Tugas Akhir ini tidak lepas dari dukungan, bimbingan, bantuan, serta arahan dari berbagai pihak. Oleh karena itu, melalui kesempatan ini Penulis mengucapkan terima kasih kepada:

1. Dr. Adian Fatchur Rochim, S.T., M.T. selaku Ketua Departemen Teknik Komputer Universitas Diponegoro yang telah memimpin Departemen Teknik Komputer.
2. Dania Eridani, S.T., M.Eng. selaku dosen pembimbing I yang telah memberikan saran serta bimbingan dalam pembuatan Tugas Akhir.

Submitted: 18 November; Accepted: 22 November; Published: 30 November

3. Adnan Fauzi, S.T., M.Kom. selaku dosen pembimbing II yang telah memberikan saran serta bimbingan dalam pembuatan Tugas Akhir.
4. Kedua orang tua yang selalu mendukung dan penulis dalam menyelesaikan Tugas Akhir.
5. Elvitro Gumelar Agung selaku *partner* Penulis yang membantu dalam pembuatan Tugas Akhir dalam pembuatan *game*.

DAFTAR PUSTAKA

[1] Togelius, J. & Champandard, A.J. & Lanzi, Pier Luca & Mateas, M. & Paiva, A. & Preuss, Mike & Stanley, K.O. (2013). Procedural content generation: goals, challenges and actionable steps. Dagstuhl Follow-Ups. 6. 61-75.
[2] Smith, Gillian. “An Analog History of Procedural Content Generation.” FDG (2015).
[3] Babin, Mathias Paul. “A Hybrid Approach to Procedural Dungeon Generation.” (2020).
[4] B. M. F. Viana and S. R. dos Santos, "A Survey of Procedural Dungeon Generation," 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (2019).
[5] J. K. Haas, “A History of the Unity Game Engine”, Worcester Polytechnic Institute, 2014.
[6] R. A. Krisdiawan, “IMPLEMENTASI PENGEMBANGAN SISTEM GDLS DAN ALGORITMA LINEAR CONGRUENTIAL GENERATOR PADA GAME PUZZLE”, Kuningan: Fakultas Ilmu Komputer Universitas Kuningan, 2018.
[7] R. Rido, dan W, Yani. “Game Development Life Cycle Guidelines”, Bandung : School of Electrical Engineering and Informatics, Institut Teknologi Bandung, 2013.
[8] Bates, Bob. “Game Design”. Boston, MA: Thomson Course Technology, 2004.

Submitted: 18 November; Accepted: 22 November; Published: 30 November