BAB 2
LANDASAN TEORI

2.1 Procedural Content Generation

Procedural content generation merupakan proses pembuatan konten secara acak melalui penggunaan algoritma [8]. Ini berbeda dari metode pembuatan konten tradisional karena sifatnya yang otomatis. Daripada membuat konten secara manual, *procedural content generation* mengimplementasikan algoritma-algoritma yang dapat mengeksekusi kode untuk menghasilkan konten. Dengan ini, hasil konten dibuat secara prosedural tanpa keterlibatan langsung oleh seorang pengembang. Hasil konten yang dibuat tergantung pada algoritma dan parameter yang dipilih.

Procedural content generation dapat digunakan untuk beberapa aspek berbeda pada sebuah *game*. Termasuk dalam hal yang dapat dihasilkan adalah tekstur, audio, lingkungan *game*, level *game*, dan lainnya [9]. Konten yang dihasilkan dapat merupakan sesuatu yang diperlukan untuk sebuah *game* dapat dijalankan seperti lingkungan *game*. Atau, konten yang dihasilkan dapat bersifat opsional dan melengkapi. Proses pembuatan konten prosedural juga dapat dilakukan saat *game* dijalankan atau saat proses pengembangan *game* [10].

Ada banyak keuntungan yang datang dengan penggunaan *procedural content generation*. Biaya dan waktu yang diperlukan untuk pengembangan sebuah *game* dapat dikurangi dengan menggunakan *procedural content generation*. Ini memungkinkan *games* untuk dibuat dengan tim lebih kecil dan skala yang lebih besar. Selain itu, *procedural content generation* meningkatkan variasi dalam pengalaman pemain. Ini membuat *game* dengan konten prosedural untuk dapat dimainkan berulang kali tanpa membosankan pemain [11].

2.2 Wave Function Collapse

Wave function collapse merupakan sebuah algoritma berbasis *constraint* yang dapat digunakan dalam *procedural content generation* [12]. Terinspirasi oleh sebuah konsep fisika, *wave function collapse* berkerja dengan mendefinisikan elemen-elemen yang akan digunakan untuk generasi prosedural seperti *tiles* serta aturan *constraint* bagi elemen bersebelahan. Aturan *constraint* digunakan untuk mengatur elemen-elemen yang dapat bersebelahan. *Wave function collapse* lalu

diimplementasikan dalam sebuah *game space* untuk penempatan elemen seperti sebuah sistem *grid*.

Wave function collapse berkerja dalam beberapa tahap. Awalnya, setiap posisi dimana sebuah elemen dapat diletakan akan memiliki peluang untuk mendapatkan setiap elemen yang telah didefinisikan sebelumnya. Saat sebuah elemen terpilih, aturan *constraint* yang telah diatur akan digunakan untuk menentukan elemen-elemen apa yang dapat diletakan pada lokasi bersebelahan. Ini akan menurunkan jumlah elemen yang mungkin diletakan pada lokasi-lokasi tersebut. Penurunan ini juga akan mempengaruhi lokasi-lokasi bersebelahan lokasi sebelumnya dan seterusnya hingga seluruh *game space* terpengaruhi. Setelah itu sebuah lokasi akan diturunkan menjadi sebuah elemen dan prosesnya berulang hingga seluruh *game space* telah terisi [13].

Ada beberapa cara yang dapat digunakan untuk menurunkan sebuah lokasi menjadi sebuah elemen. Satu cara yang dapat digunakan adalah dengan memilih elemen secara acak dari semua pilihan elemen yang tersedia. Alternatif dari metode ini adalah untuk memberi setiap elemen sebuah *weight* yang mempengaruhi probabilitas terpilihnya elemen tersebut [7]. Cara pemilihan lokasi yang akan diturunkan akan berbeda berdasarkan cara mana yang diimplementasikan. Jika *weights* tidak digunakan, maka lokasi dengan jumlah pilihan elemen terkecil akan terpilih. Jika *weights* digunakan, maka nilai shannon entropy setiap lokasi dihitung dan lokasi dengan *shannon entropy* terkecil akan terpilih [14]. Rumus 2.1 menunjukkan cara perhintungan *Entropy*.

$$Entropy = -\sum p(x) * log p(x) \tag{2.1}$$

2.3 Tilemaps

Tilemaps merupakan sebuah teknik yang sudah lama diterapkan dalam pengembangan *game* [15]. Teknik ini melibatkan penggunaan komponen dasar yang disebut *tile* yang dapat berupa gambar, *sprite*, model 3D, dan lainnya. *Tiles* yang digunakan disusun dan digunakan berulang kali dalam sebuah struktur *grid* untuk membuat sebuah lingkungan *game*. *Tiles* tersebut lalu dapat digabungkan dengan logika *game* untuk menerapkan fitur-fitur seperti *collision* atau *pathfinding*.

Tilemaps dapat diimplmentasikan dalam berbagai cara berbeda. *Tilemaps* dapat digunakan untuk membuat sebuah lingkungan yang bersifat 2D, 3D, atau

isometrik. Selain itu, *tiles* yang digunakan dapat memiliki bentuk-bentuk berbeda seperti bentuk kotak atau heksagon. *Tiles* juga dapat diletakan pada *layers* berbeda untuk meningkatkan kualitas visual dan mengimplementasikan logika *game*.

2.4 Divide and Conquer

Divide and conquer merupakan sebuah strategi pemrograman yang digunakan untuk mempercepat dan memudahkan penyelesaian tugas yang besar dan rumit. Dengan *divide and conquer*, sebuah tugas akan dibagi menjadi beberapa tugas lebih kecil. Tugas-tugas lebih kecil ini kemudian diselesaikan satu per satu dan hasilnya digabung untuk mendapatkan solusi akhir [16]. Tugas sering dibagi secara rekursif saat mengimplementasikan *divide and conquer*. Hal ini berarti tugas akan terus dibagi menjadi tugas yang lebih kecil dan lebih kecil lagi hingga tugas menjadi mudah untuk diselesaikan. Selain itu, tugas dapat dibagi secara manual dan disimpan dalam sebuah struktur data seperti sebuah lis atau *array*.

Divide and conquer sering digunakan karena keuntungan-keuntungan yang didaptkan dengan implementasinya. Keuntungan utama yang didapatkan adalah pengurangan waktu yang dibutuhkan untuk menyelesaikan sebuah tugas. Hal ini dapat dicapai oleh *divide and conquer* karena efisiensi algoritma yang lebih tinggi. Dengan membagi sebuah tugas kompleks menjadi subtugas yang lebih kecil, pekerjaan yang perlu dilakukan menjadi lebih mudah dan cepat untuk diselesaikan [17]. Dan, *divide and conquer* dapat digunakan untuk memanfaatkan arsitektur paralel dengan menghasilkan subtugas yang bersifat independen. Dengan ini, banyak subtugas dapat diselesaikan secara serentak untuk mempercepat penyelesaian tugas utama [18].

2.5 Chunks

Dalam konteks *map* pada *game*, sebuah *map* dapat dibagi menjadi potongan-potongan bernama *chunks*. *Chunks* merupakan potongan atau bagian dari sebuah *map* dengan ukuruan yang tetap [19]. Semua *chunks* pada sebuah *game* bergabung untuk membentuk *map* secara keseluruhan. Penggunaan *chunks* merupakan sebuah teknik yang sering digunakan dalam pengembangan *game*. Satu contoh *game* populer yang menggunakan sistem ini adalah *Minecraft* dimana *map* terdiri atas *chunks* dengan ukuran 16x16x256 blok [20].

Pengunaan *chunks* terutama digunakan untuk untuk meningkatkan performa

dari *game*. Dengan *chunks*, *game* dapat dioptimisasi dengan hanya fokus terhadap *chunks* yang dekat dengan pemain. *Chunks* yang jauh dari pemain dapat ditampilkan dengan detail rendah atau tidak ditampilkan sama sekali. Teknik ini menghemat sumber daya karena hanya sebagian dari *map* perlu diproses. Selain itu, *chunks* dapat digunakan dengan *procedural content generation* untuk membangun *map* yang terasa tak terbatas. Hal ini dilakukan dengan membangun *chunks* saat pemain mendekatinya. Dengan ini, *map* akan terus dibangun dan berkembang selama pemain sedang memainkan *game* dengan sistem ini [21].