BAB 3
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian

3.1.1 Identifikasi Masalah

Identifikasi masalah dilakukan untuk mengidentifikasi masalah-masalah utama yang ingin dijadikan fokus dibalik penelitian yang akan dilaksanakan. Masalah ditemukan saat melakukan studi literatur mengenai implmentasi algoritma *wave funtion collapse* untuk melakukan *procedural content generation*. Ditemukan bahwa waktu yang diperlukan algoritma untuk membangun *map* meningkat secara signifikan saat ukuran *map* diperbesar. Dengan ini, terpikirkan untuk mengimplementasikan *divide and conquer* pada proses pembangunan *map* dan membandingkan dampaknya.

3.1.2 Studi Literatur

Studi literatur dilakukan untuk mengumpulkan informasi mengenai topik-topik relevan yang berhubungan dengan penelitian ini. Topik-topik tersebut antara lain *procedural content generation*, *wave function collapse*, *tilemaps*, *divide and conquer*, dan *chunks*. Informasi yang didapatkan melalui pendalaman topik-topik tersebut lalu digunakan untuk membantu melaksanakan penelitian ini.

3.1.3 Perancangan Aplikasi

Perancangan aplikasi meliputi proses desain dibalik *procedural content generation* untuk pembangunan *map* 2D yang ingin dilakukan. Perancangan secara utama dilakukan dengan membuat *flowchart* untuk merencanakan cara implementasi algoritma *wave function collapse* dan *divide and conquer* untuk proses pembangunan *map*.

3.1.4 Pembangunan Aplikasi

Pembangunan aplikasi meliputi implementasi hasil perancangan untuk membangun aplikasi untuk penelitian ini. Termasuk dalam hal ini adalah implementasi algoritma *wave function collapse* dan *divide and conquer* untuk

menghasilkan konten secara prosedural saat aplikasi dijalankan pengguna. Aplikasi dibangun menggunakan *game engine* Unity.

3.1.5 Testing dan Evaluasi Aplikasi

Testing dilakukan untuk menguji hasil pembangunan aplikasi yang telah diperoleh. Proses ini melibatkan *white box testing*. Dengan ini, setiap fitur pada aplikasi diuji untuk melihat apakah dapat berjalan dengan benar atau tidak. *Debugging* lalu dilakukan untuk memperbaiki *error* atau *bug* yang ditemukan. Evaluasi dilakukan dengan menjalankan aplikasi dan merekam waktu yang diperlukan oleh aplikasi untuk membangun *map* secara prosedural. Aplikasi dijalankan dengan berbagai parameter berbeda seperti ukuran *map* dan jumlah *chunks* yang digunakan. Setiap set parameter akan dijalankan sepuluh kali untuk mendapatkan nilai yang konsisten.

3.1.6 Analisis Hasil Evaluasi Aplikasi

Hasil evaluasi dianalisa untuk melihat efek dari implementasi *divide and conquer* pada waktu pembangunan *map* pada aplikasi. Ini dilakukan dengan pertama menghitung waktu rata-rata dari setiap set parameter yang dijalankan. Rata-rata tersebut lalu dibandingkan untuk melihat perbedaan hasilnya dan menentukan apakah implementasi *divide and conquer* memberi dampak positif pada proses pembangunan *map* prosedural.

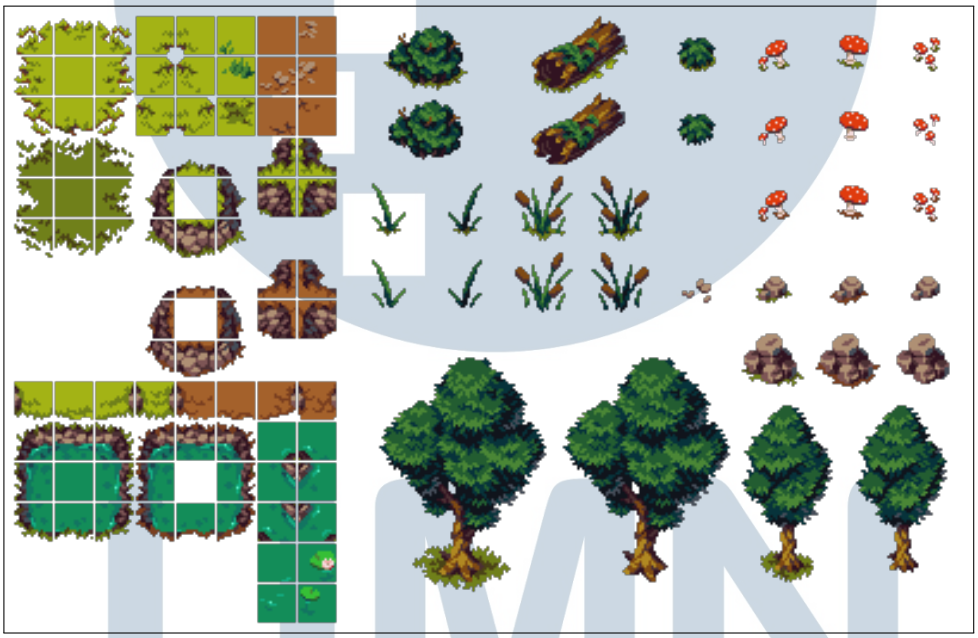
3.1.7 Penulisan Laporan

Sebuah laporan penelitian ditulis yang mencakupi seluruh proses penelitian dari dilakukannya studi literatur hingga analisis evaluasi pengguna. Penulisan laporan dilakukan dengan bantuan dan arahan dari doesen pembimbing Wirawan Istiono, S.Kom., M.Kom

3.2 Perancangan Aplikasi

Aplikasi yang dibangun merupakan sebuah sistem yang mengimplementasikan *divide and conquer* pada algoritma *wave function collapse* untuk membuat *map* 2D secara prosedural. *Map* dibangun dalam bentuk sebuah *grid* bentuk persegi. *Grid* tersebut terdiri atas objek-objek yang bernama *cell*.

Sebuah *cell* merupakan representasi sebuah area pada *map* [22]. Setiap *cell* mengandung berbagai informasi seperti bobot, entropi, dan pilihan *tiles* yang dapat digunakan. *Tiles* adalah apa yang digunakan untuk membangun *map* dan memberi tampilannya. *Tiles* yang digunakan berupa gambar 2D dan dapat dilihat pada Gambar 3.1. Pilihan *tiles* mengacu kepada *tiles* yang mempunyai peluang untuk muncul pada lokasi sebuah *cell*. *Cells* dapat mempunyai lebih dari satu pilihan *tiles* namun proses pembangunan *map* akan menurunkan pilihan ini menjadi hanya satu *tile* untuk setiap *cell*. *Divide and conquer* diimplementasikan pada algoritma *wave function collapse* dengan membagi *grid* awal menjadi beberapa *subgrid* bernama *chunks*. Setiap *chunk* dibangun satu per satu untuk mendapatkan hasil *map* akhir.

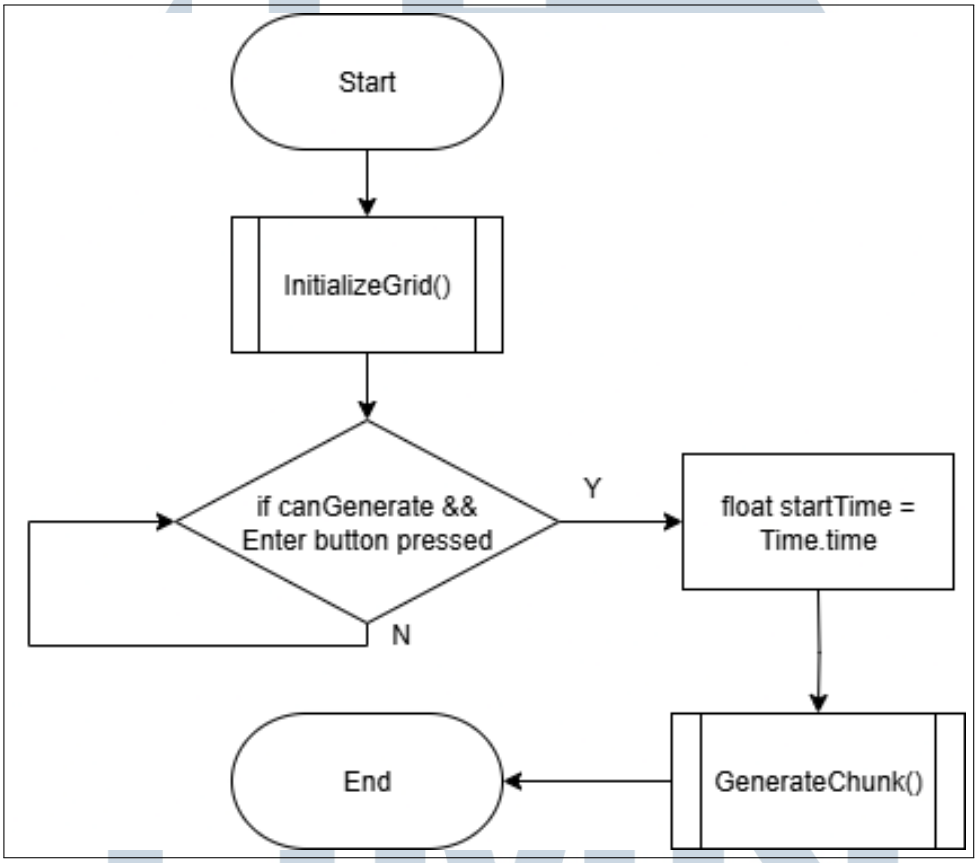


Gambar 3.1. ”Free Topdown Fantasy - Forest - Pixelart Tileset” by aamatniekss

3.2.1 Flowchart Wave Function Collapse

Proses *procedural content generation* dimulai dengan membangun sebuah *grid* 2D yang akan digunakan untuk membuat *map* 2D. *Grid* tersebut terdiri atas *gameobject cell* yang mengandung beberapa informasi seperti bobot, entropi, dan pilihan *tile* yang dapat digunakan. *Grid* tersebut juga akan dibagi menjadi beberapa *subgrid chunk*. Semua ini dilakukan dalam modul *InitializeGrid*. Setelah *grid* telah dibangun, tombol *enter* dapat ditekan untuk memulai pembangunan *map* secara prosedural. Tahap pertama adalah untuk merekam waktu saat proses pembangunan *map* dimulai. Hal ini dilakukan dengan memanggil fungsi bernama *Time.time* yang

disediakan oleh Unity. Fungsi ini akan memberi nilai waktu dalam detik sejak dimulainya aplikasi dan nilai tersebut akan disimpan pada sebuah variabel bernama *startTime*. Selanjutnya, *map* akan digenerasi pada modul *GenerateChunk* secara bertahap dengan membangun setiap *chunk* satu per satu. *Flowchart* dari proses generasi *map* dapat dilihat pada Gambar 3.2.

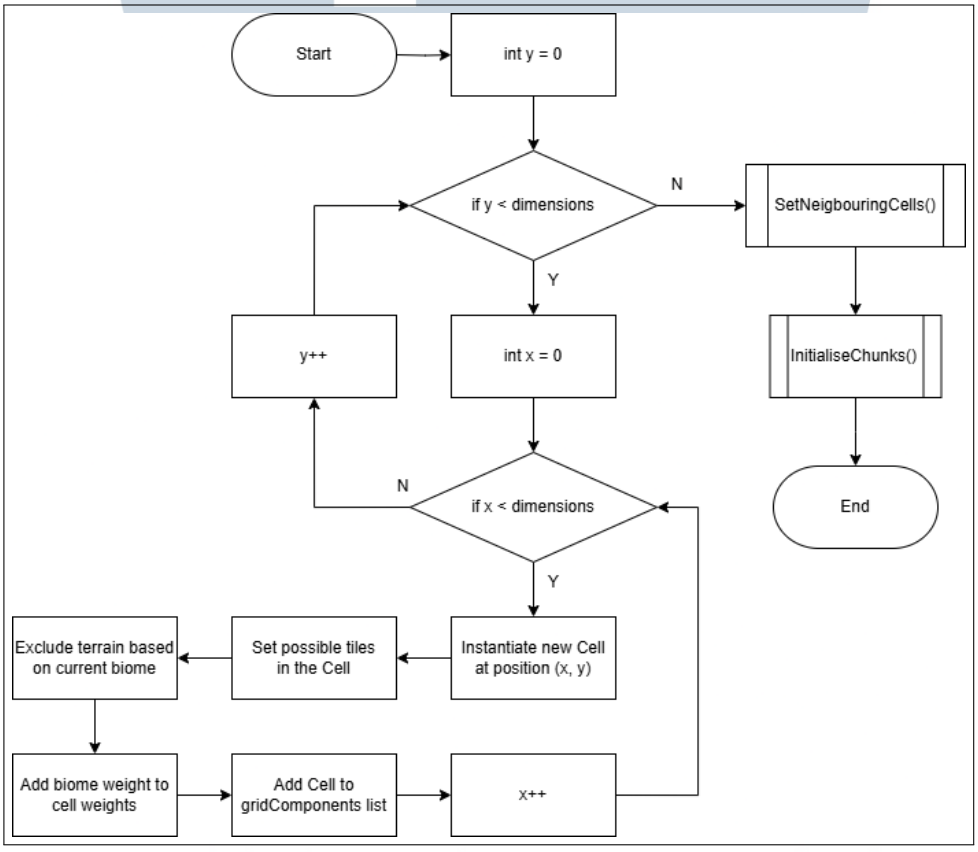


Gambar 3.2. Flowchart *Wave Function Collapse*

3.2.2 Flowchart Modul InitializeGrid

Grid yang dibangun merupakan sebuah *grid* 2D dalam bentuk persegi. Panjang dari sisi persegi tersebut ditentukan oleh sebuah variabel bernama *dimensions*. Pembangunan *grid* dimulai dengan menggunakan sebuah *nested for-loop* untuk menginkrementasi dua buah integer yaitu x dan y. *For-loop* pertama menginkrementasi nilai y dan untuk memulai *for-loop* kedua. *For-loop* kedua berfungsi untuk menginkrementasi nilai x dan untuk membuat *cells* pada *grid*. Pada setiap iterasi dari *for-loop* kedua, sebuah *cell* baru akan dibuat dan nilai variabel x dan y akan digunakan untuk menentukan posisi *cell* tersebut.

Selanjutnya, *cell* yang telah dibuat akan diatur berdasarkan bioma yang sedang digunakan. Pilihan *tiles* yang dapat digunakan akan diberi pada *cell*, *terrain* tertentu akan dikecualikan, dan bobot bioma akan ditambahkan. *Cell* tersebut akan kemudian dimasukan dalam sebuah lis yang mengandung semua *cell* bernama *gridComponents*. Lis tersebut digunakan untuk merepresentasikan *grid* yang sedang dibangun. Setelah semua *cell* telah dibuat dan diatur, *cell* tetangga setiap *cell* akan dicari dan disimpan pada setiap *cell* untuk memudahkan pencarian pada tahap-tahap selanjutnya menggunakan modul *SetNeighbouringCells*. *Grid* juga dibagi menjadi beberapa *subgrid* untuk mengimplementasikan fitur-fitur *chunks* menggunakan modul *InitialseChunks*. *Flowchart* modul *InitializeGrid* dapat dilihat pada Gambar 3.3.

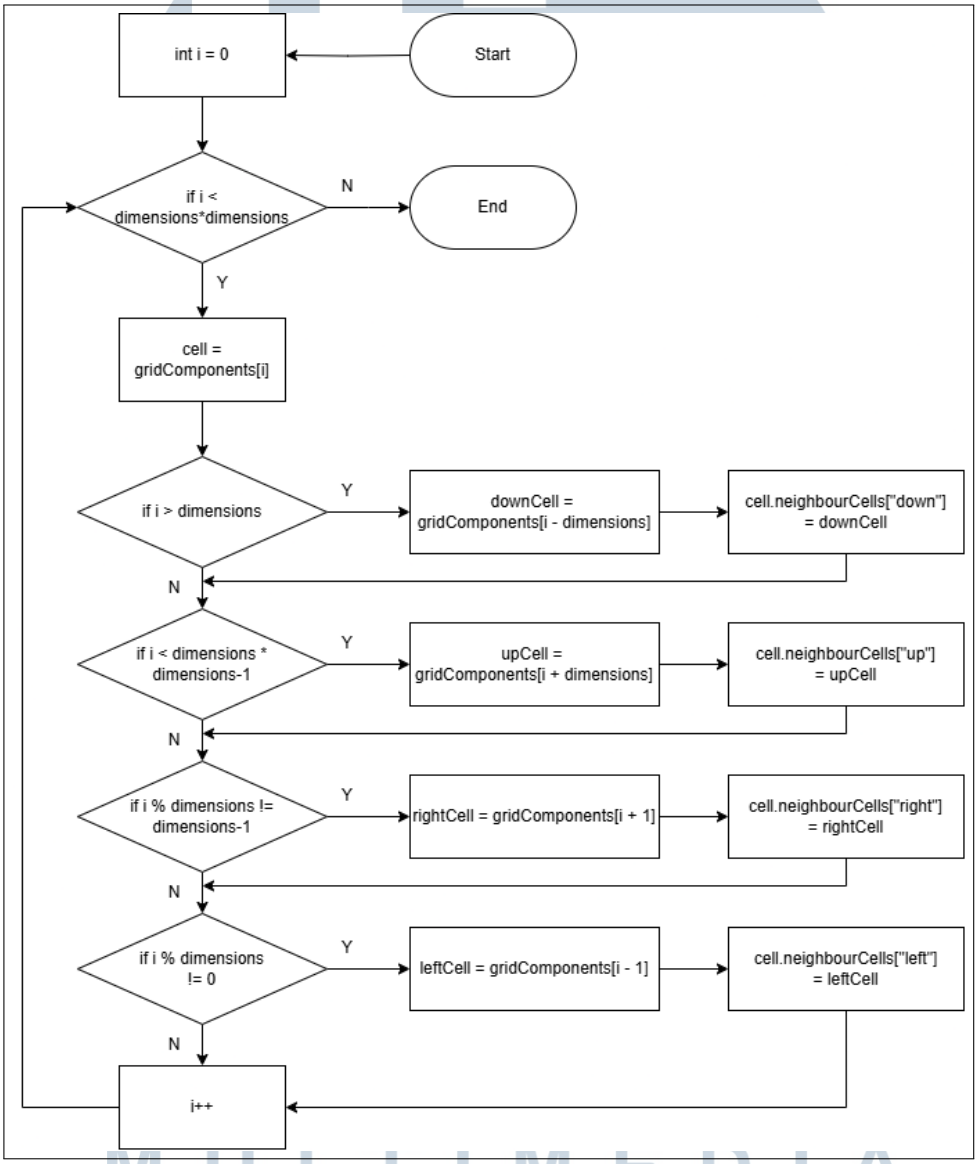


Gambar 3.3. Flowchart Modul *Initialize Grid*

3.2.3 Flowchart Modul SetNeighbouringCells

Modul *SetNeighbouringCells* mengecek setiap *cell* pada lis *gridComponents* untuk melihat *cell* tetangga yang dimiliki setiap *cell*. Hal ini dilakukan menggunakan sebuah *for-loop* untuk menginkremen sebuah variabel i. Variabel

i digunakan sebagai indeks untuk mengakses komponen lis *gridComponents*. Pengecekan lalu dilakukan berdasarkan lokasi sebuah *cell* untuk menentukan *cell* tetangga yang tersedia. Tetangga-tetangga yang ditemukan lalu dimasukan dalam sebuah *dictionary* milik *gameobject cell* yang bernama *neighbourCells*. *Flowchart* modul *SetNeighbouringCells* dapat dilihat pada Gambar 3.4.

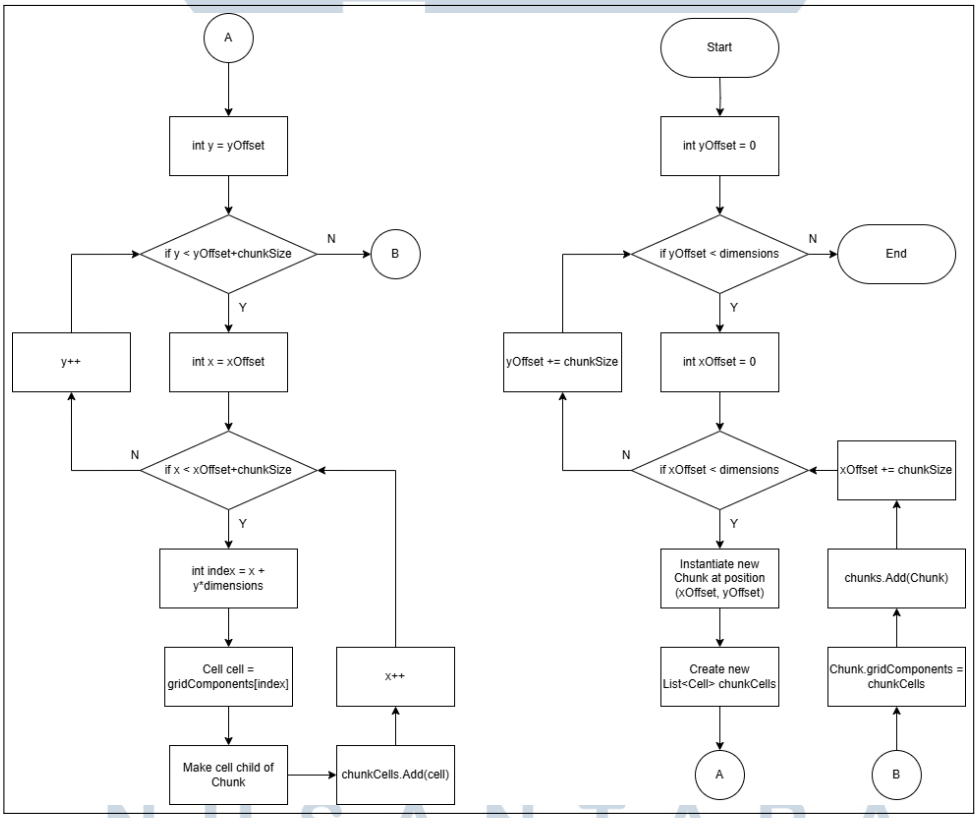


Gambar 3.4. Flowchart Modul *Set Neighbouring Cells*

3.2.4 Flowchart Modul InitialiseChunks

Modul *InitialiseChunks* digunakan untuk membagi *grid* menjadi beberapa *subgrid* yang disebut *chunk*. *Chunks* digunakan pada proses pembangunan *map*

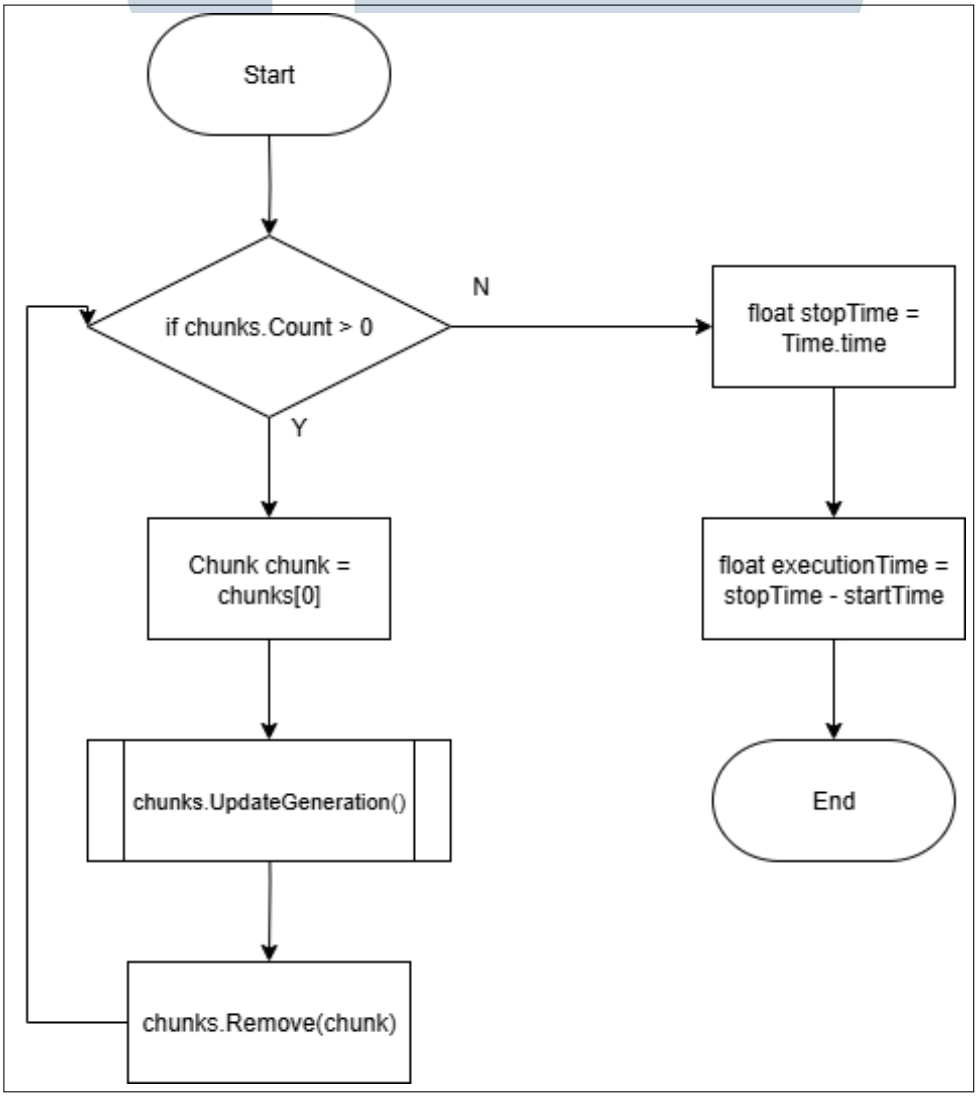
untuk dapat melakukan generasi secara bertahap dan mengurangi propogasi bobot yang perlu dilakukan. Tahap pertama dalam membuat sebuah *chunk* adalah untuk menjalankan dua buah *for-loops* untuk menginkrementasi nilai xOffset dan yOffset. Kedua nilai ini berguna untuk mengatur posisi *chunk* yang dibuat. Ukuran dari *chunk* ditentukan oleh variabel *chunkSize*. Selanjutnya, sebuah lis *cells* baru dibuat untuk menampung semua *cells* yang akan menjadi bagian *chunk* yang telah dibuat. Dua *for-loops* lagi digunakan untuk menemukan *cells* pada lis *gridComponents* yang perlu dimasukan ke *chunk*. Indeks dari *cells* tersebut ditemukan dengan menginkrementasikan variabel integer x dan y serta menggunakan nilai xOffset dan yOffset dari *for-loop* sebelumnya. *Cells* yang berhasil ditemukan akan dimasukan ke lis *cells* milik *chunk*. Saat lis tersebut telah selesai dibangun, lisnya akan dimasukan ke dalam varibel lokal *gridComponents* milik *chunk* dan *chunk* akan dimasukan dalam sebuah lis yang mengandung semua *chunk* yang telah dibuat. *Flowchart* modul *InitialiseChunks* dapat dilihat pada Gambar 3.5.



Gambar 3.5. Flowchart Modul *Initialise Chunks*

3.2.5 Flowchart Modul GenerateChunk

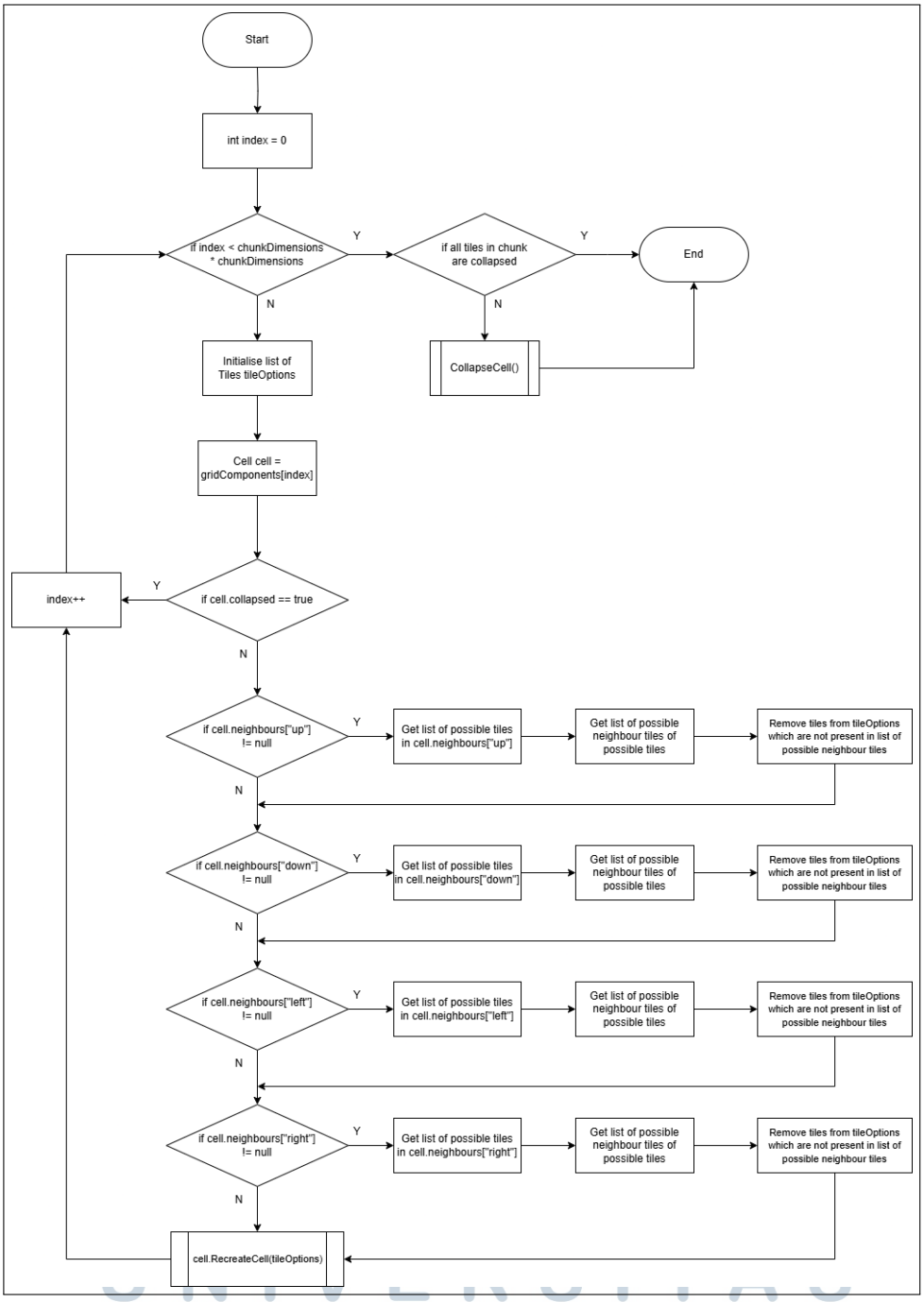
Modul *GenerateChunk* digunakan untuk memulai proses pembangunan *map* bertahap per *chunk*. Pertama, sebuah *chunk* akan diambil dari lis yang mengandung semua *chunks* yang telah dibuat. Modul *UpdateGeneration* lalu dipanggil pada *chunk* tersebut untuk memulai proses generasi pada *chunk*. Setelah *chunk* telah selesai dibangun chunk tersebut akan dihapus dari lis. Proses ini terus diulangi hingga isi lis *chunks* kosong. Jika lis *chunks* sudah tidak ada isi maka semua *chunks* telah selesai dibangun dan waktu akan direkam lagi menggunakan fungsi *Time.time* dan disimpan pada variabel *stopTime*. Waktu eksekusi algoritma lalu akan dihitung dengan mengurangi nilai *stopTime* dengan nilai *startTime*. *Flowchart* modul *GenerateChunk* dapat dilihat pada Gambar 3.6.



Gambar 3.6. Flowchart Modul *Generate Chunk*

3.2.6 Flowchart Modul UpdateGeneration

Modul *UpdateGeneration* digunakan untuk melakukan proses propogasi dimana pilihan *tiles* pada setiap *cell* pada sebuah *chunk* akan diperbarui berdasarkan pilihan *tiles* pada *cell* tetangga. Modul dimulai dengan menjalankan sebuah *for-loop* yang digunakan untuk mengakses setiap *cell* pada sebuah *chunk* satu per satu. Sebuah lis *tileOptions* juga akan diinisialisasi yang mengandung semua *tiles* yang dapat digunakan dalam pembangunan *map*. Saat sebuah *cell* terpilih, yang pertama dilakukan adalah untuk mengecek apakah *cell* tersebut sudah *collapsed*. Jika iya, berarti sebuah *tile* sudah terpilih untuk *cell* tersebut dan proses propogasi tidak diperlukan. Jadi, *for-loop* akan lanjut ke *cell* selanjutnya. Jika tidak, maka proses propogasi akan berjalan. Setiap *cell* tetangga yang ada akan diakses. Dari *cell* tetangga diambil semua pilihan *tiles* yang ada. Setelah itu, sebuah lis akan dibangun yang mengandung semua *tiles* yang dapat bersebelahan dengan pilihan *tiles* milik *cell* tetangga. Perbandingan lalu dilakukan antara lis ini dengan lis *tileOptions* yang telah diinisialisasi pada awal *for-loop*. Semua *tiles* pada *tileOptions* yang tidak muncul pada lis yang telah dibangun akan dihapus dari *tileOptions*. Proses ini dilakukan untuk setiap *cell* tetangga untuk mendapatkan lis *tileOptions* akhir. Modul *RecreateCell* lalu dipanggil dengan lis *tileOptions* untuk memperbarui lis pilihan *tiles* dan nilai entropi pada *cell*. *For-loop* dijalankan hingga propogasi telah dilakukan untuk setiap *cell* yang tidak sudah *collapsed*. Setelah itu akan dicek apakah semua *cells* pada *chunk* sudah bersifat *collapsed*. Jika iya, maka *chunk* telah selesai dibangun. Jika tidak, modul *CollapseCell* akan dipanggil untuk *collapse* sebuah *cell* dengan nilai entropi terendah. *Flowchart* modul *UpdateGeneration* dapat dilihat pada Gambar 3.7.

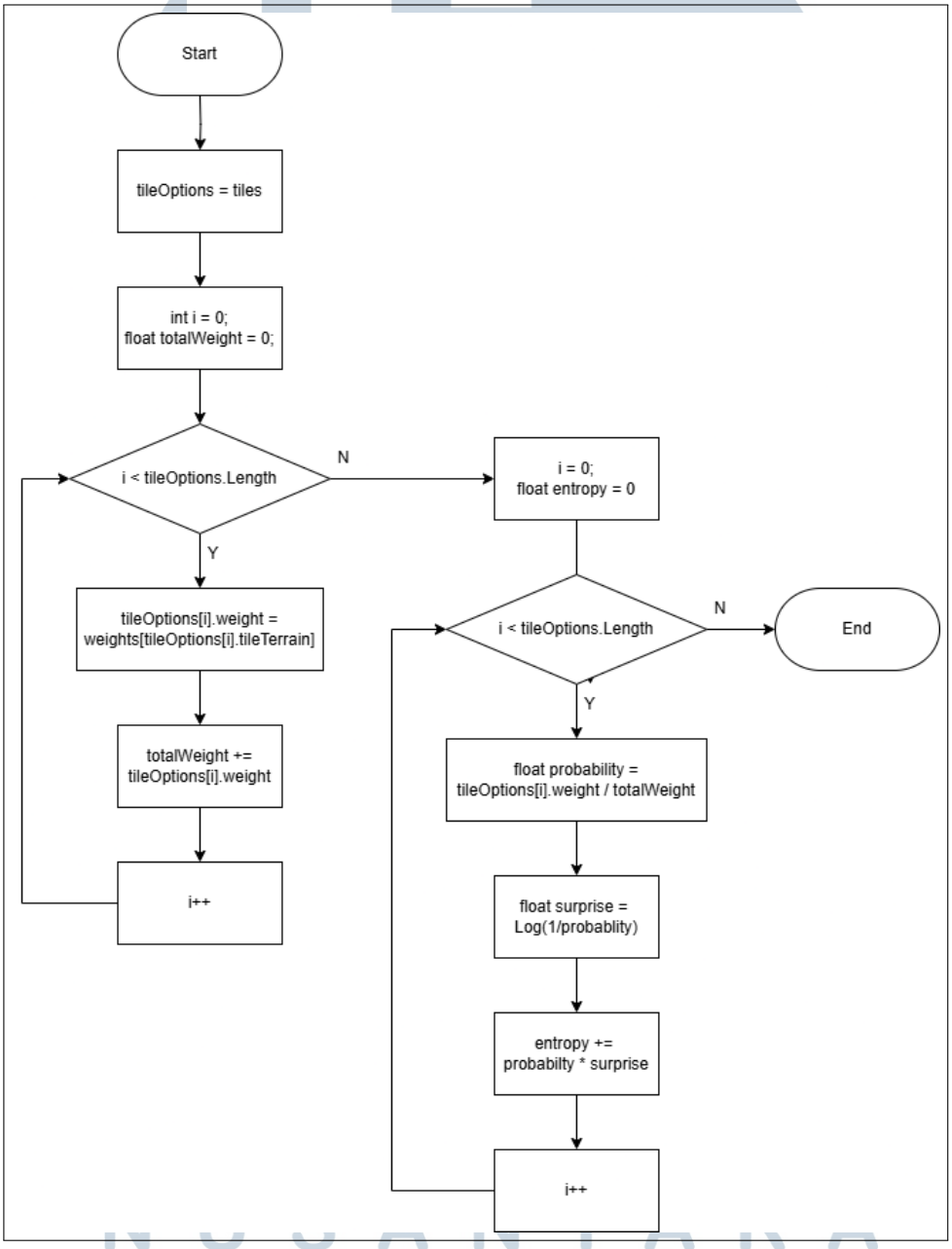


Gambar 3.7. Flowchart Modul *Update Generation*

3.2.7 Flowchart Modul RecreateCell

Modul *RecreateCell* mulai dengan memperbarui lis pilihan *tiles* pada *cell*. Lalu, nilai entropi dari *cell* akan dihitung. Hal ini dilakukan dengan pertama menghitung jumlah dari bobot setiap pilihan *tile* pada *cell* menggunakan sebuah

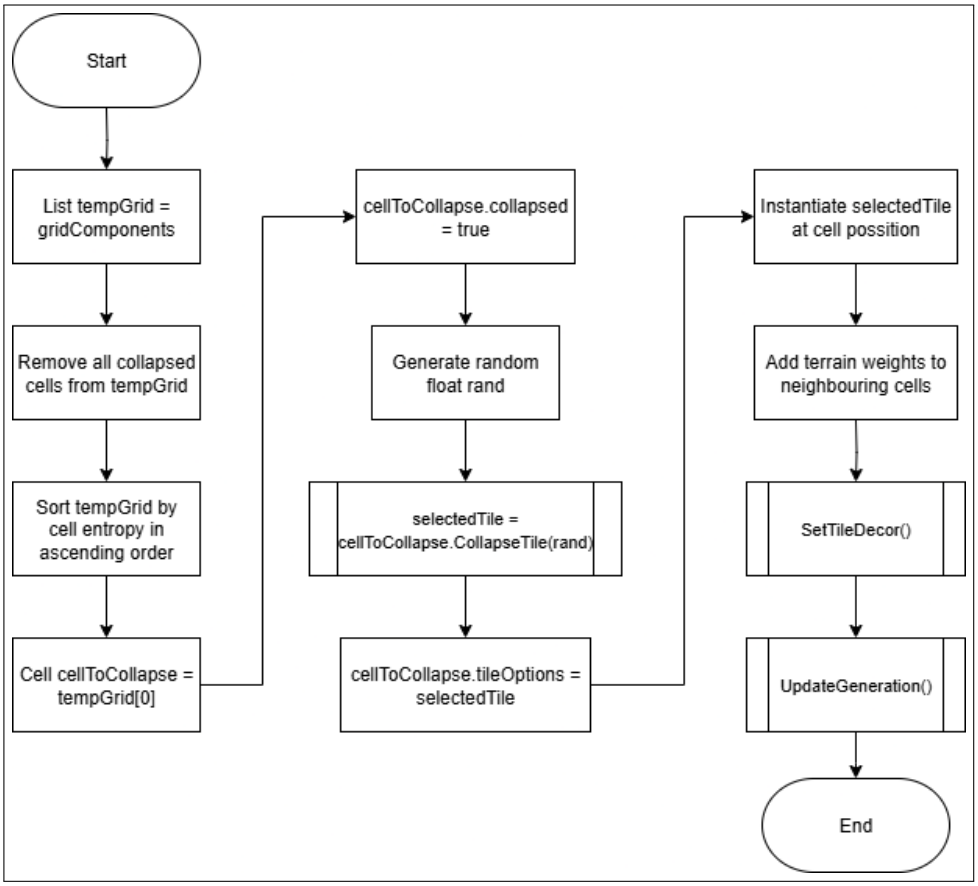
for-loop. Setelah itu, *for-loop* kedua digunakan untuk menghitung nilai entropi dari setiap pilihan *tile*. Entropi dihitung dengan mengkalikan probabilitas *tile* tersebut terpilih dengan nilai *surprise* terpilihnya *tile* tersebut. Semua nilai entropi *tile* pilihan lalu dijumlahkan dan menjadi nilai entropi *cell*. *Flowchart* modul *RecreateCell* dapat dilihat pada Gambar 3.8.



Gambar 3.8. Flowchart Modul *Recreate Cell*

3.2.8 Flowchart Modul CollapseCell

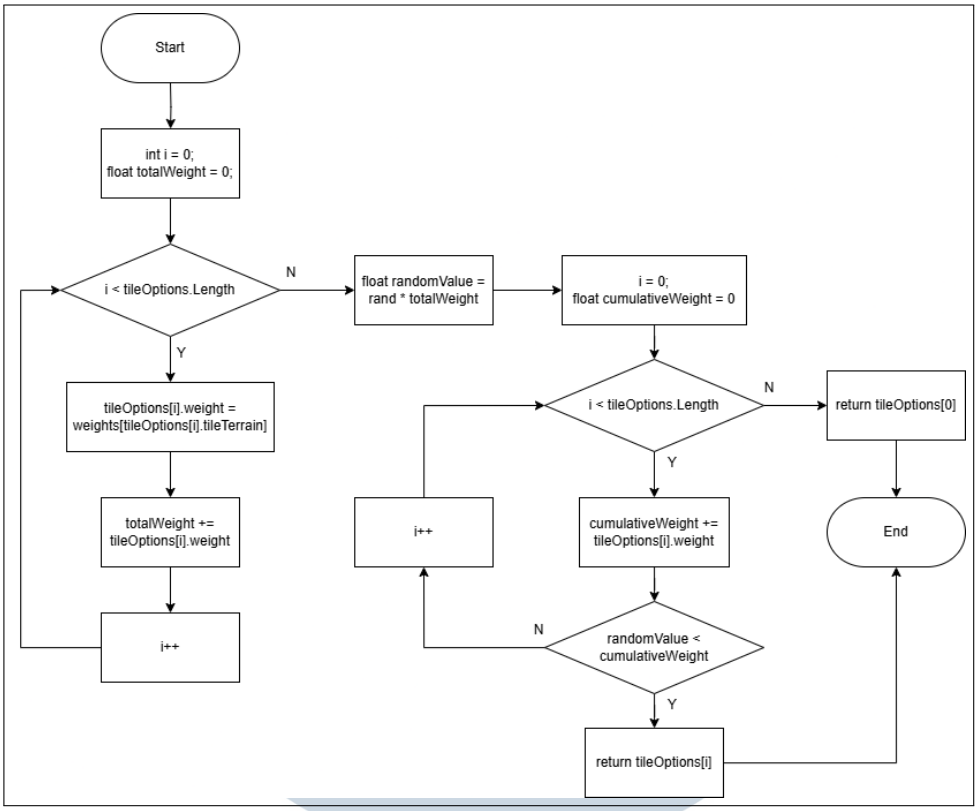
Modul *CollapseCell* digunakan untuk sebuah *cell* pada *chunk* dengan nilai entropi terendah untuk di *collapse*. Saat sebuah *cell* di *collapse*, lis pilihan *tiles* pada *cell* yang dapat berisi banyak pilihan akan diturunkan menjadi satu pilihan. Modul dimulai dengan membuat sebuat lis baru bernama *tempGrid* yang merupakan sebuah salinan dari lis *gridComponents* milik *chunk*. Semua *cell* yang bersifat *collapsed* lalu akan dihapus dari lis *tempGrid*. Sisa *cells* pada lis lalu dirutkan secara urutan menaik berdasarkan nilai entropi *cell* masing-masing. Dengan ini *cell* dengan nilai entropi terendah yang diberi nama *cellToCollapse* dapat ditemukan. Setelah ini adalah proses untuk *collapse cell* tersebut. Pertama, variabel *collapsed* pada *cell* diberi nilai *true* untuk menandakan bahwa *cell* sudah *collapsed*. Setelah itu, sebuah *float* acak akan dibuat untuk pemilihan *tiles*. *float* tersebut memiliki nilai antara 0 sampai 1 dan digunakan dalam modul *CollapseTile* untuk memilih sebuah *tile*. Beberapa hal dilakukan dengan *tile* yang didapatkan. Lis *tileOptions* milik *cell* akan diperbarui untuk hanya mengandung *tile* yang dipilih. Hal ini dilakukan karena *cell* masih dapat diperlukan sebagai *cell* tetangga pada proses propogasi pada modul *UpdateGeneration*. *Tile* yang terpilih juga akan dibuat pada lokasi *cell*. Dan, bobot untuk nilai *terrain* dari *tile* yang terpilih akan ditambahkan pada lis bobot-bobot *cell* tetangga dari *cell* yang di *collapse*. Modul *SetTileDecor* lalu dipanggil untuk menempatkan dekorasi pada *tile* yang telah ditempatkan. Dekorasi yang tersedia pada *tileset* yang digunakan berupa gambar-gambar bertema hutan seperti pohon, semak, dan lainnya. Setelah *cell* berhasil di *collapse* maka modul *UpdateGeneration* akan dipanggil untuk melanjutkan proses pembangunan *chunk*. *Flowchart* modul *CollapseCell* dapat dilihat pada Gambar 3.9.



Gambar 3.9. Flowchart Modul *Collapse Cell*

3.2.9 Flowchart Modul CollapseTile

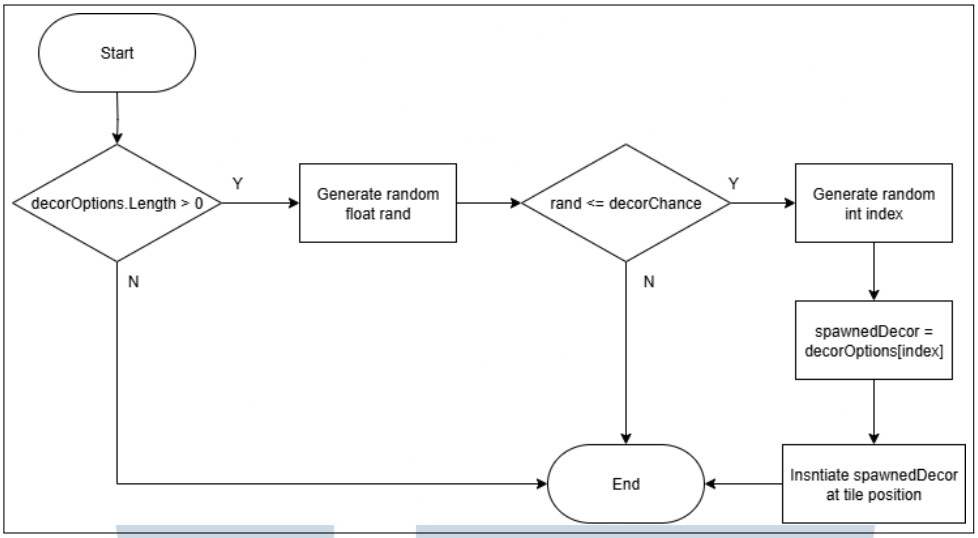
Modul *CollapseTile* digunakan untuk sebuah *tile* dari lis pilihan *tiles* secara acak menggunakan metode *roulette selection*. Modul berawal dengan menjalankan sebuah *for-loop* untuk menghitung total dari bobot-bobot semua *tiles* pada lis pilihan *tiles* milik *cell*. Setelah itu, sebuah *float* bernama *randomValue* didapatkan dengan mengkali nilai *rand* yang didapatkan dari modul *CollapseCell* dengan nilai total bobot *tiles*. *Float rand* memiliki nilai antara 0 sampai 1 jadi variabel *randomValue* akan memiliki nilai antara 0 sampai total bobot *tiles*. Dengan nilai *randomValue*, proses *roulette search* dapat dimulai menggunakan sebuah *for-loop*. *For-loop* tersebut akan mengakses setiap *tiles* pada lis pilihan satu per satu dan menambahkan ke sebuah variabel bernama *cumulativeWeight*. Jika sebuah *tile* membuat nilai *cumulativeWeight* menjadi lebih besar dibandingkan *randomValue*, maka *tile* tersebut akan terpilih. Pilihan *default* dari tahap ini adalah *tile* pertama pada lis pilihan. *Flowchart* modul *CollapseTile* dapat dilihat pada Gambar 3.10.



Gambar 3.10. Flowchart Modul *Collapse Tile*

3.2.10 Flowchart SetTileDecor

Modul *SetTileDecor* digunakan untuk menempatkan sebuah gambar dekorasi pada *tile* yang telah ditempatkan. Setiap *tile* akan mempunyai sebuah lis bernama *decorOptions* yang berisi semua gambar dekorasi yang dapat muncul pada *tile*. Jika lis tersebut tidak kosong, maka sebuah *float* acak bernama *rand* akan dibuat dan dibandingkan dengan sebuah float bernama *decorChance*. Variabel *decorChance* merupakan sebuah *float* dengan nilai antara 0 dan 1 yang merepresentasikan peluang sebuah *tile* untuk memiliki gambar dekorasi. Jika nilai *rand* lebih kecil dibandingkan nilai *decorChance* maka gambar dekorasi akan dibuat. Hal ini dilakukan dengan mendapatkan sebuah int acak bernama *index* untuk memilih sebuah gambar dekorasi dari lis *decorOptions* secara acak. Gambar dekorasi yang terpilih lalu dibuat pada lokasi *tile*. *Flowchart* modul *SetTileDecor* dapat dilihat pada Gambar 3.11.



Gambar 3.11. Flowchart Modul *Set Tile Decor*

3.3 Perancangan Eksperimen Evaluasi

Eksperimen evaluasi dirancang dengan tujuan untuk mengukur dampak penggunaan *divide and conquer* pada waktu eksekusi algoritma *wave function collapse*. Hal tersebut dilakukan dengan merekam waktu yang dibutuhkan algoritma untuk membangun *map* saat tidak menggunakan *divide and conquer* dan saat menggunakan *divide and conquer*. Perekaman waktu dilakukan dengan merekam waktu algoritma mulai dijalankan dan waktu algoritma selesai membangun *map*. Dengan ini, waktu eksekusi algoritma dapat dihitung dengan mengurangi waktu selesai algoritma dengan waktu mulai algoritma [23] [24]. Perekaman waktu dilakukan oleh kode yang telah ditulis dalam bahasa C# dan dijalankan oleh *game engine* Unity.

$$ExecutionTime = EndTime - StartTime \tag{3.1}$$

Ada beberapa variabel yang digunakan pada eksperimen yang dirancang. Variabel dependen yang digunakan adalah waktu eksekusi algoritma. Variabel ini adalah apa yang akan direkam dan dianalisa. Variabel independen yang digunakan adalah ukuran *map* dan jumlah *chunks* yang digunakan. Kedua variabel tersebut adalah variabel yang akan dirubah nilainya untuk melihat pengaruhnya pada nilai waktu eksekusi algoritma. Variabel kontrol pada eksperimen ini adalah pilihan *tiles* serta nilai *seed* yang diberi kepada *pseudo random number generator* yang

digunakan. Nilai kedua variabel tersebut tidak diganti untuk memastikan bahwa hasil *map* yang dibangun konsisten.

Proses pembangunan *map* dijalankan sebanyak sepuluh kali untuk setiap kombinasi nilai ukuran *map* dan jumlah *chunks*. Rata-rata dari waktu yang terekam lalu dihitung untuk mendapatkan nilai waktu eksekusi akhir. Hal ini dilakukan karena waktu eksekusi algoritma dapat bervariasi akibat faktor-faktor seperti kondisi perangkat keras, proses latar belakang, dan lainnya. Dengan menghitung nilai rata-rata, waktu eksekusi yang lebih akurat dapat didapatkan [25].