BAB III

METODOLOGI PENELITIAN DAN PERANCANGAN SISTEM

3.1. Metodologi Penelitian

Metodologi penelitian yang diterapkan dalam pengaplikasian fractal noise dalam WFC untuk terrain generation berbasis tile adalah sebagai berikut:

a. Studi Literatur

Dalam tahap studi literatur, teori yang bersangkutan dengan topik skripsi dipelajari. Teori yang dipelajari adalah *WFC, Simple Tiled Model, Perlin Noise, Fractal Noise,* dan *Tilemap*.

b. Perancangan Game

Dalam tahap ini, game yang akan dibuat dirancang dalam bentuk *Game Design Document* (GDD). GDD menjabarkan konsep dari game secara ringkas.

c. Pembangunan Game dan Implementasi

Dalam tahap ini, dilakukan pembuatan game menggunakan *game engine* Unity dan pembuatan asset model menggunakan *software* Blender sesuai dengan rancangan GDD.

d. Pengujian Game

Dalam tahap pengujian, game akan diuji ke sejumlah responden yang dipilih secara acak. Responden tersebut akan mengisi kuisioner seusai pengujian game tersebut.

e. Evaluasi

Dalam tahap ini, hasil kuisioner dianalisa untuk mendapatkan sebuah

kesimpulan. Kuisioner menggunakan pertanyaan yang disediakan dari *Game User Experience Satisfaction Scale* (GUESS).

3.2. Perancangan Aplikasi

Perancangan aplikasi dibagi menjadi dua bagian, yaitu Game Design Document, *flowchart*, dan *mockup* aplikasi.

3.2.1. Game Design Document

1. Story

   Ken mempunyai berbagai macam tile dengan jumlah yang sangat banyak. Oleh karena itu, dia ingin membuat sebuah dunia yang luasnya tak terhingga. Untuk membuat dunia tersebut, Ken menggunakan algoritma Fractal Perlin Noise dan Wave Function Collapse.

2. Formal Elements

   a) Player

      Single Player: *Game* ini dimainkan oleh satu orang.

      Player vs Game: Pemain melakukan eksplorasi pada *terrain* yang dibentuk oleh algoritma.

   b) Objective

      Player melakukan eksplorasi pada terrain yang telah dibentuk.

   c) Procedures

      Pemain diberikan tampilan menu utama. Pemain dapat memasukkan nilai *seed* ke dalam input box untuk mendapatkan satu dunia spesifik. Player dapat menekan tombol *generate* untuk memulai *game*.

Proses dari pembentukan terrain akan ditampilkan saat tombol *generate* ditekan.

Karakter pemain akan muncul setelah pembentukan dunia selesai dan pemain dapat mengeksplorasi dunia menggunakan karakter tersebut.

Pemain dapat kembali ke menu utama dan mencoba mengeksplorasi dunia baru atau menekan tombol *quit* untuk keluar dari *game*.

d) Rules

Pemain tidak bisa melakukan eksplorasi bila terrain belum dibentuk.

Karakter pemain tidak dapat berjalan melewati pohon dan akan melambat bila berjalan melewati laut.

Karakter pemain dipengaruhi oleh gravitasi.

e) Resources

Pemain dapat menggunakan tombol keyboard WASD untuk menggerakkan karakter pemain.

f) Conflict

Pemain harus melakukan eksplorasi pada terrain yang telah dibentuk.

g) Boundaries

Pemain memiliki jarak pandang yang terbatas dari perspektif *top down* yang digunakan.

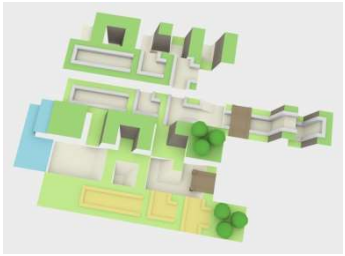
h) Outcome

Pemain telah mengeksplorasi terrain yang dibentuk algoritma.

3. Art Asset

Asset berupa model 3D yang digunakan dalam game adalah sebagai berikut:

Tabel 3.1. Tabel Art Asset

Nama Asset	Sumber	Gambar
Terrain Tileset	Author: Garrison Cahyadi, 2020	
Skater (*male*)	Author: Kenney, 2019 Source: https://kenney.nl/assets/animated-characters-2	

4. Sound Asset

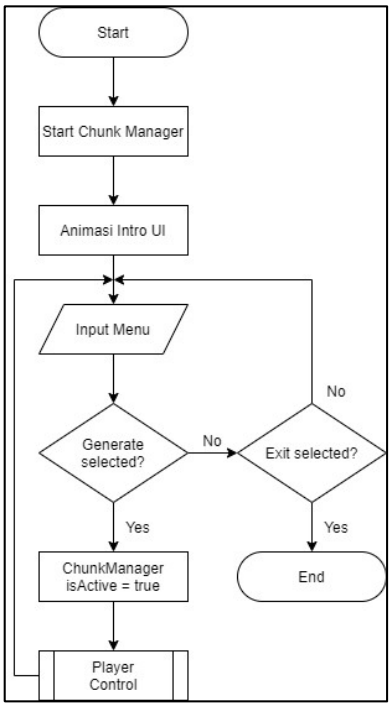
Asset suara yang digunakan dalam game adalah sebagai berikut:

Tabel 3.2. Tabel Sound Asset

Nama Asset	Sumber	Deskripsi
Clear Air	https://incompetech.com/	Musik latar belakang
Click Sound	http://soundbible.com/783-Click.html	Suara klik

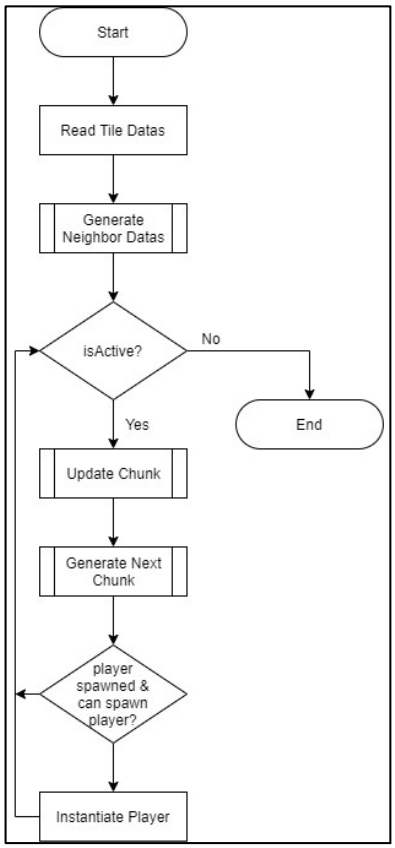
3.2.2. Flowchart

Flowchart dirancang untuk memvisualisasikan alur dari aplikasi yang akan dibuat. Berikut adalah beberapa *flowchart* dari *game* yang akan dibangun.



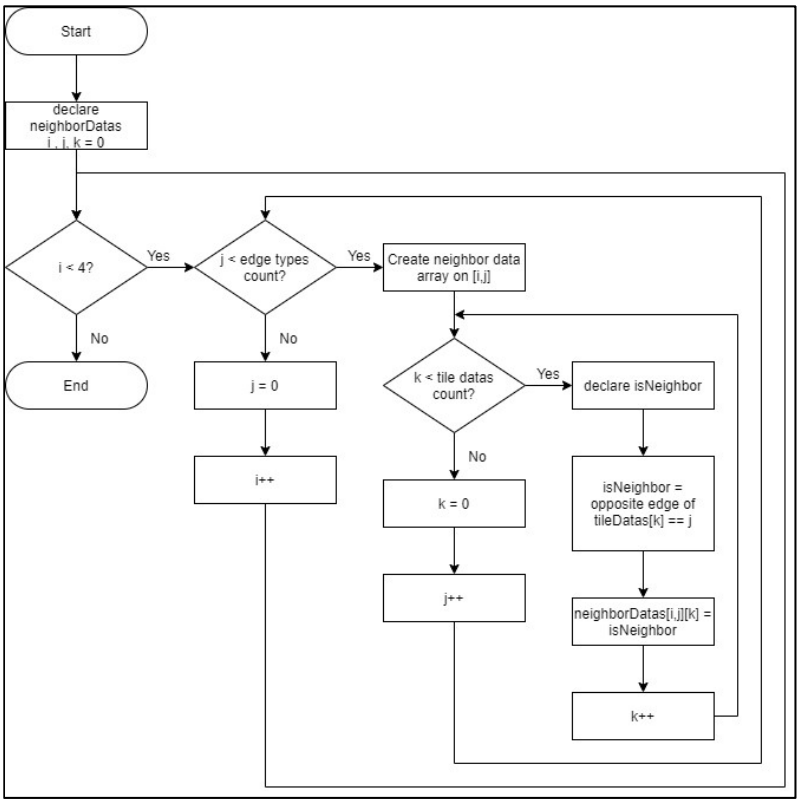
Gambar 3.1. *Flowchart* aplikasi secara umum

Gambar 3.1. adalah *flowchart* dari aplikasi yang dibuat secara umum. Pertama, objek *chunk manager* diinisialisasi. *Chunk manager* merupakan sebuah objek yang berfungsi untuk memilih chunk yang akan dibangun, dan menentukan chunk aktif untuk mengaktifkan dan menon-aktifkan objek chunk. Proses inisialisasi akan dijelaskan lebih lanjut dalam *flowchart* pada Gambar 3.2. Kemudian, animasi pembuka untuk tampilan menu dimainkan. Setelah animasi pembuka telah dimainkan, pemain dapat memilih tombol *generate*, *quit*, atau mengisi *seed* ke dalam *input box*. Bila tombol *generate* ditekan, variabel *isActive* pada objek *chunk manager* diubah menjadi *true* supaya proses pengelolaan *chunk* dapat dijalankan. Setelah terrain di sekitar posisi awal telah selesai dibentuk, karakter pemain akan di-*instantiate* dalam proses lain dan dapat dikontrol oleh pemain.



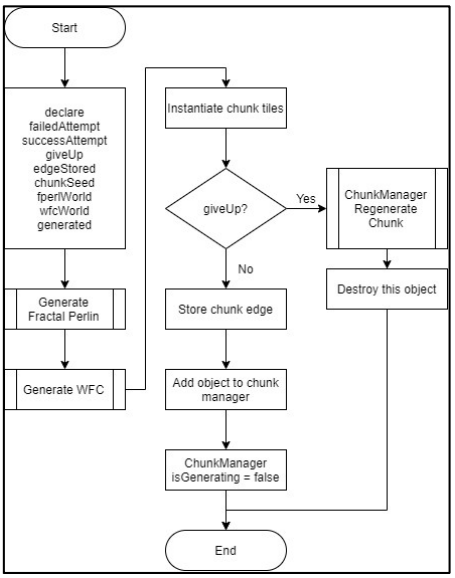
Gambar 3.2. *Flowchart* proses c*hunk manager*

Gambar 3.2. adalah *flowchart* dari c*hunk manager*. Pertama, *tile data* yang telah dibuat dalam Unity dibaca dan dimuat ke dalam objek chunk manager. *Tile data* merupakan *data* berupa *ScriptableObject* yang menyimpan *model* yang digunakan, rotasi objek, *edge identifier*, dan bobot kemunculan. Data tersebut kemudian digunakan untuk membentuk *neighbor data,* proses ini akan dijelaskan lebih lanjut di *flowchart* dari Gambar 3.3. *Chunk manager* akan memroses *chunk* saat variabel *isActive* diubah menjadi *true* (melalui tombol *generate* di menu). *Chunk Manager* akan meng-*instantiate* karakter pemain setelah terrain selesai dibentuk dan objek pemain itu sendiri belum di-*instantiate*.



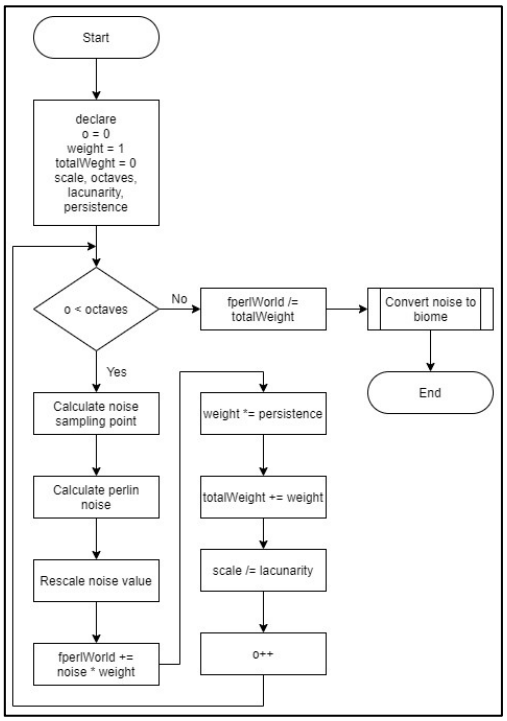
Gambar 3.3. *Flowchart* pembentukan *neighbor data*

Gambar 3.3. merupakan *flowchart* dari proses pembentukan *neighbor data*. Dalam proses ini, variabel *neighborDatas* dipopulasi dengan sekumpulan indeks *tile* (berupa *array boolean*) yang memiliki *edge* bernilai indeks j pada arah yang berlawanan dari indeks i.



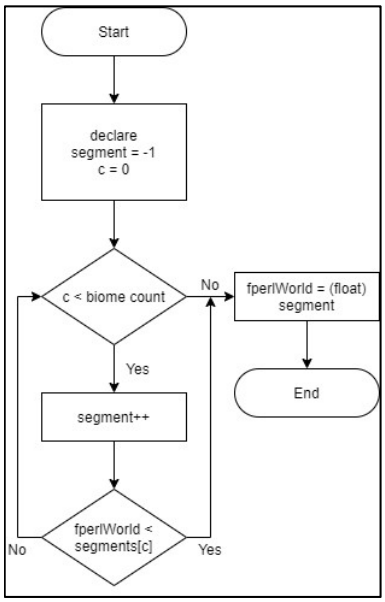
Gambar 3.4. *Flowchart chunk generator*

Gambar 3.4. merupakan *flowchart* yang menjelaskan proses generasi chunk secara umum. Objek c*hunk generator* melakukan inisialisasi variabel terlebih dahulu. Algoritma *fractal perlin noise* dijalankan setelah inisialisasi. Hasil dari *fractal perlin noise* kemudian digunakan pada algoritma *wave function collapse*. Setelah kedua algoritma tersebut dijalankan, objek tile di-*instantiate* berdasarkan indeks *tile* yang dimiliki setiap cell (kecuali di beberapa chunk pertama dimana objek tile diinstantiate dalam setiap iterasi dari *wave function collapse* untuk visualisasi proses). Bila proses generasi berhasil (diindikasikan dari variabel *giveUp* bernilai *false*), indeks *tile* yang dimiliki *cell* pada tepi chunk dan objek *generator* itu sendiri disimpan ke dalam objek *chunk manager*, dan pesan selesai dikirim ke chunk manager. Sebaliknya, g*enerator* akan meninstruksikan c*hunk manager* untuk menjalankan proses regenerasi *chunk* dan objek *generator* yang gagal dihancurkan.



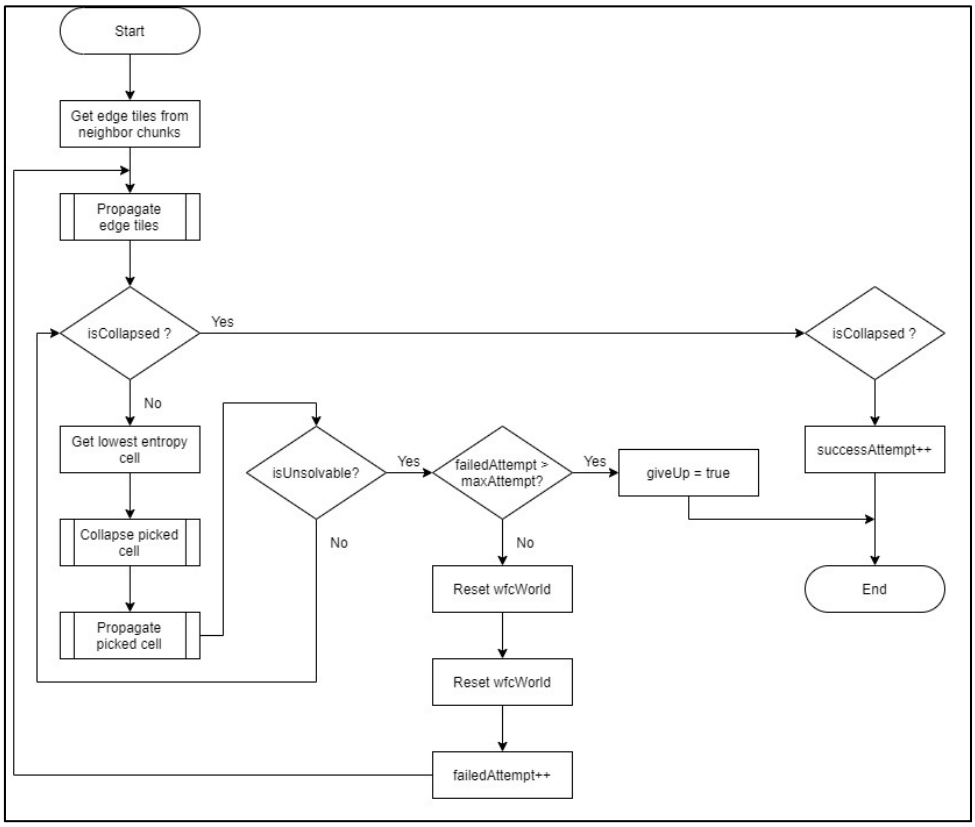
Gambar 3.5. *Flowchart fractal perlin noise*

Gambar 3.5. adalah *flowchart* dari fungsi *fractal perlin noise*. *Perlin noise* disampel pada setiap posisi *cell* sebanyak jumlah oktaf. Ukuran dari *noise* yang disampel bergantung pada variabel *scale,* dan noise tersebut diberi bobot sebesar variabel *weight*. Nilai dari *weight* kemudian ditambah ke variabel *totalWeight* yang akan digunakan untuk normalisasi *fractal noise*, dimana *noise* yang dihasilkan memiliki *range* 0-1. Pada setiap iterasi, ukuran dari noise yang disampel pada iterasi selanjutnya diperkecil berdasarkan nilai variabel *lacunarity* dan bobot dari sampel selanjutnya berkurang berdasarkan nilai variabel *persistence*. Hasil dari akumulasi *noise* dinomalisasikan, kemudian dikonversi ke nilai biome.



Gambar 3.6. *Flowchart* konversi *noise* ke nilai *biome*

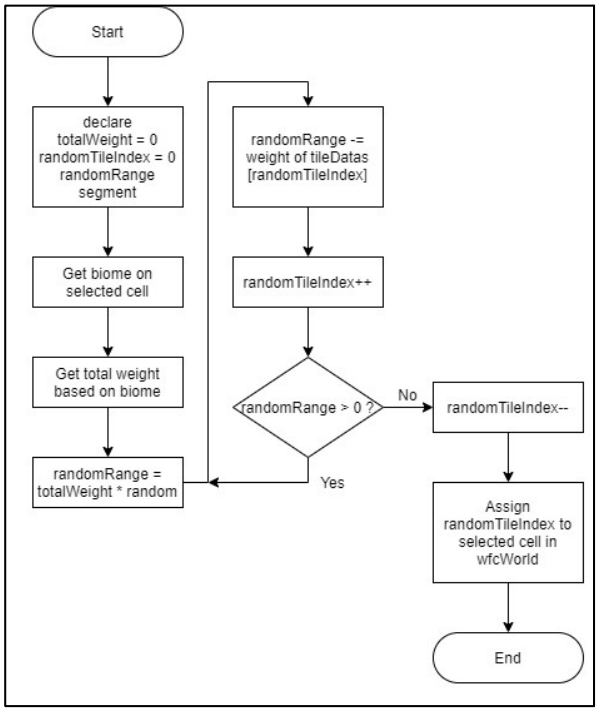
Gambar 3.6. menampilkan *flowchart* dari konversi *fractal perlin noise* ke nilai *biome* (dinamai sebagai *segment* dalam kode). Nilai biome didapatkan melalui *looping* dimana variabel *segment* di-*increment*, lalu dilakukan perbandingan nilai antara *fractal perlin noise* dengan salah satu nilai dari *segments* secara berurutan. Variabel *segments* merupakan *array* yang menyimpan batas atas dari setiap *biome*. Nilai *biome* didapatkan bila salah satu nilai *segments melebihi* nilai *noise* atau *looping* telah selesai dilakukan.



Gambar 3.7. *Flowchart wave function collapse*

Gambar 3.7. merupakan *flowchart* dari fungsi *wave function collapse*. *Tile data* yang tersimpan dalam *cell* yang bersebelahan dengan tepi chunk diambil terlebih dahulu untuk melakukan propagasi awal supaya *chunk* yang dihasilkan terhubung dengan *chunk* yang telah dibuat. Kemudian, algoritma ini menjalankan sebuah *loop* sampai setiap *cell* dalam *chunk* hanya memiliki satu *tile* (dikenal sebagai fase *collapsed*). *Loop* ini terdiri dari pemilihan *cell* dengan entropi paling rendah, meng-*collapse* dan mempropagasi *cell* yang dipilih, kemudian dilakukan pengecekan bila *chunk* tersebut *unsolveable* dimana terdapat sebuah *cell* yang tidak memiliki tile. Pemilihan *cell* dengan entropi paling rendah dilakukan dengan menghitung nilai entropi pada setiap *cell* untuk menentukan nilai entropi yang

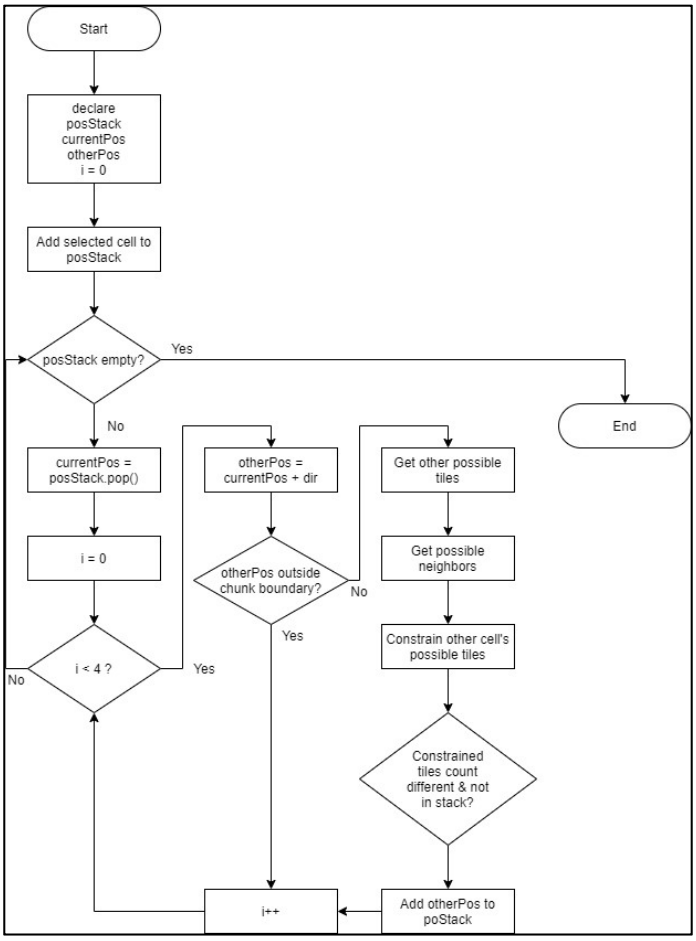
paling rendah, kemudian nilai entropi tersebut dibandingkan dengan nilai entropi pada setiap cell. Saat jumlah *cell* yang memiliki entropi paling rendah lebih dari satu, salah satu cell tersebut akan dipilih secara acak. Bila chunk tersebut *unsolvable*, WFC diulang dari awal bila nilai *failedAttempt* masih di bawah atau sama dengan *totalAttempt*. Saat nilai *failedAttempt* melewati *totalAttempt*, *generator* tersebut akan mengubah nilai variabel *giveUp* menjadi *true* dan fungsi WFC selesai. Bila chunk selalu tidak *unsolvable* dan telah *collapsed*, variabel *successAttempt* di-*increment* dan fungsi WFC selesai.



Gambar 3.8. *Flowchart* fungsi *collapse* dari WFC

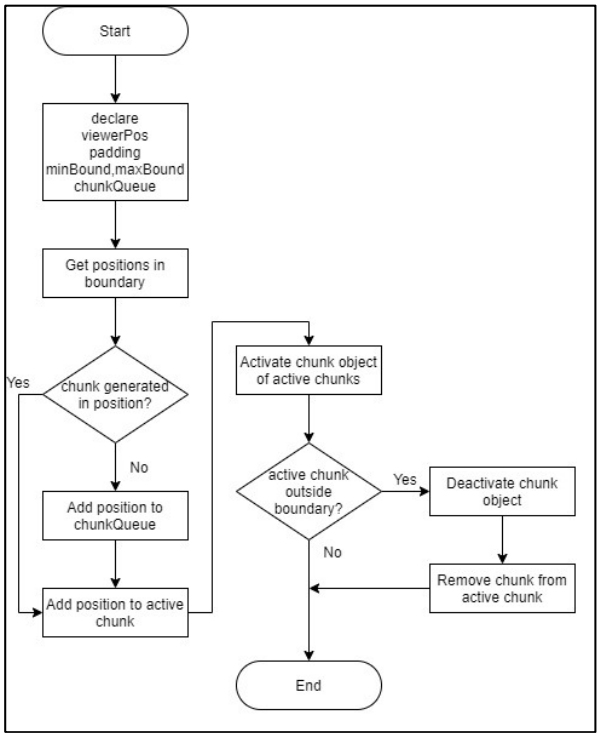
Gambar 3.8. adalah *flowchart* dari fungsi *collapse* dari WFC. Fungsi ini digunakan untuk memilih satu *tile* secara acak dari *tile* yang dimiliki *cell* berentropi paling rendah. *Tile* dipilih menggunakan sistem *weighted random*

number generator. Setiap bobot dari *tile* (dipengaruhi oleh *biome* dari *fractal perlin noise*) dijumlah ke variabel *totalWeight*. Sebuah angka *random* dari 0 ke *totalWeight* dimasukkan ke variabel *randomRange*. Kemudian, dilakukan iterasi dengan variabel *randomTileIndex* sebagai iterator. Nilai dari *randomRange* dikurangi dari bobot tile yang dimiliki cell satu demi satu. Bila nilai *randomRange* tidak lebih dari 0, indeks tile yang dipilih secara *random* ditemukan. Tile yang tidak terpilih dibuang sehingga *cell* tersebut hanya menyimpan satu *tile*.



Gambar 3.9. *Flowchart* fungsi *propagate* dari WFC

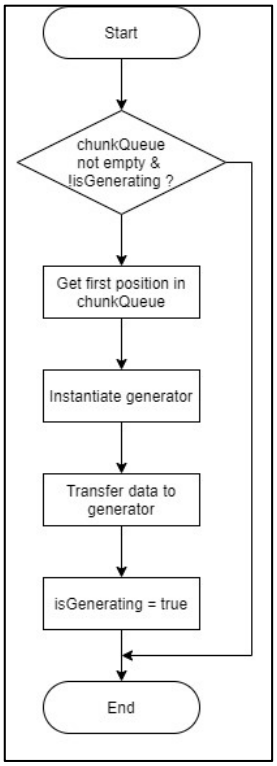
Gambar 3.9. merupakan *flowchart* dari fungsi *propagate* dari WFC. Fungsi ini beroperasi pada *stack* yang menyimpan *cell* yang akan dipropagasi. Pada setiap *cell* yang dipropagasi, *tile* yang dimiliki *cell* yang bersebelahan di-*constrain* dengan gabungan dari *tile* yang didapatkan dari *neighbor data* berdasarkan *edge* dari setiap *tile* yang dimiliki *cell* yang dipropagasi sambil mempertimbangkan arahnya. *Cell* yang bersebelahan kemudian dimasukkan ke dalam *stack* bila mengalami perubahan jumlah *tile* dan tidak berada di dalam *stack*.



Gambar 3.10. *Flowchart* fungsi *update chunk* dari *chunk manager*

Gambar 3.10. adalah *flowchart* dari fungsi u*pdate chunk* yang dimiliki c*hunk manager*. Pertama, posisi *chunk* yang termasuk dalam sebuah *boundary* diambil. *Boundary* ini ditentukan dari posisi karakter pemain (bila ada, posisi 0,0 digunakan bila karakter pemain tidak ada) dan variabel *view distance* yang

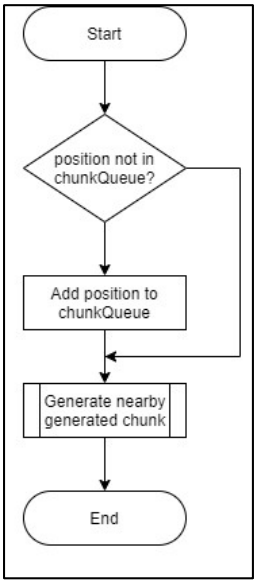
ditentukan dalam *chunk manager*. Posisi yang telah diambil dimasukkan ke dalam *list active chunk*. Posisi tersebut juga dimasukkan ke dalam *chunkQueue* (variabel berupa *linked list* tetapi dioperasikan seperti *queue* dalam kondisi normal) dengan prioritas berdasarkan posisi *chunk* bila terrain belum terbentuk dalam *chunk* tersebut. Setiap *chunk* yang aktif kemudian dicek apabila mereka berada di luar *boundary*. *Chunk* aktif yang di luar *boundary* akan dinonaktifkan dan dihapus dari *list active chunk*.



Gambar 3.11. *Flowchart* fungsi *generate next chunk* dari *chunk manager*

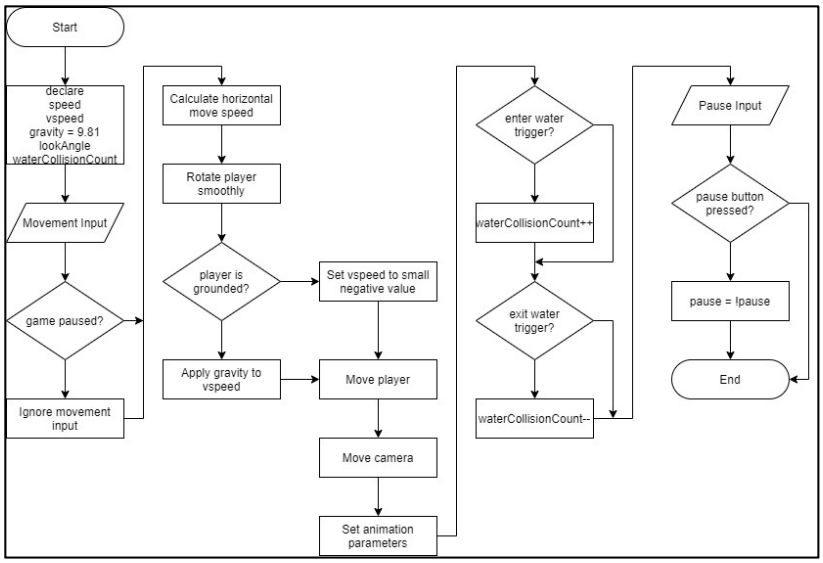
Gambar 3.11. adalah *flowchart* dari fungsi *generate next chunk* yang dimiliki objek c*hunk manager*. Bila tidak ada *chunk* yang sedang membentuk *terrain* dan variabel *chunkQueue* tidak kosong, posisi pertama dalam *chunkQueue* diambil. Objek g*enerator* akan diinstantiate pada posisi tersebut, kemudian beberapa data

dari c*hunk manager* (seperti posisi *chunk*) dikirim ke g*enerator* dan objek c*hunk manager* mengganti nilai dari variabel *isGenerating* menjadi *true* supaya c*hunk manager* tidak menginstantiate *generator* baru sebelum *generator* yang sudah ada selesai membuat terrain.



Gambar 3.12. *Flowchart* fungsi *regenerate chunk* dari *chunk manager*

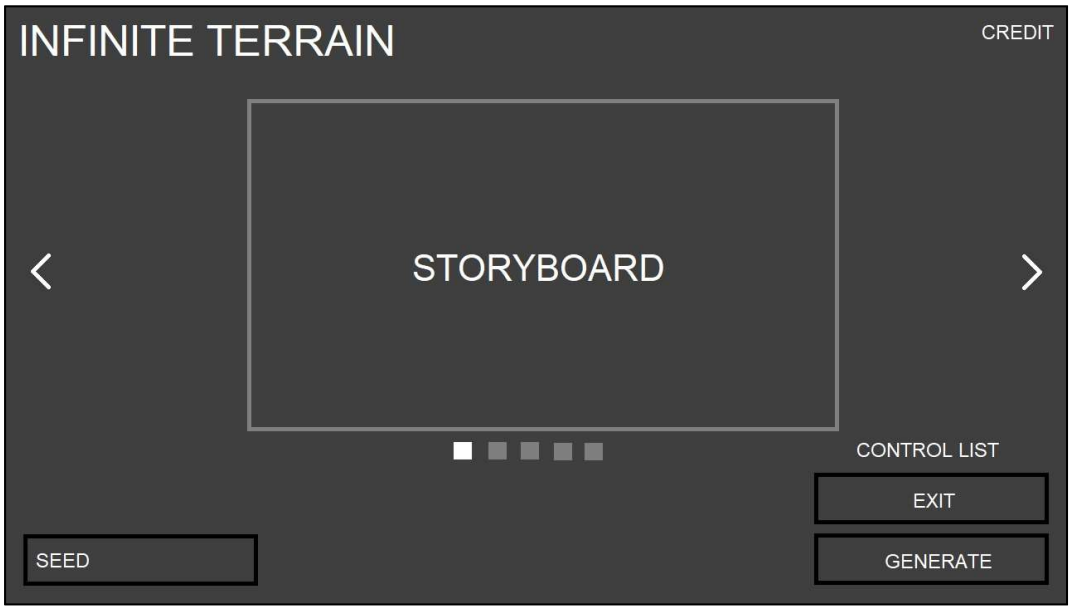
Gambar 3.12. adalah *flowchart* dari fungsi *regenerate chunk* yang dimiliki objek *chunk manager*. Posisi chunk dari objek *generator* yang gagal membentuk *terrain* akan dimasukkan kembali ke *chunkQueue* di depan. Kemudian, salah satu *generator* yang telah berhasil membentuk terrain dan bersebelahan dengan *generator* gagal dipilih untuk membentuk ulang terrain.



Gambar 3.13. *Flowchart* karakter pemain

Gambar 3.13. adalah *flowchart* dari karakter pemain. Pemain dapat melakukan *input* gerakan yang kemudian diproses. *Input* gerakan diabaikan bila *game* sedang di-*pause*. Karakter akan digerakkan oleh komponen *CharacterController* sehingga gravitasi perlu diimplementasi secara manual melalui variabel *vspeed*. Nilai dari *vspeed* meningkat bila karakter tidak menyentuh tanah, sebaliknya *vspeed* diberi nilai konstan yang rendah bila menyentuh tanah supaya *ground checking* masih bisa dijalankan. Setelah kecepatan horizontal dan vertikal dihitung, posisi karakter dipindahkan melalui komponen *CharacterController* dan posisi kamera kemudian diatur sesuai posisi karakter ditambah *offset* tertentu. Variabel *waterCollisionCount* menyimpan jumlah objek air yang sedang tersentuh oleh karakter. Kecepatan karakter akan melambat bila nilai dari variabel *waterCollisionCount* lebih dari 0. Pemain juga dapat menginput *pause*, dimana game akan di-*pause* atau di-*unpause*.

3.2.3 Mockup Aplikasi



Gambar 3.14. Mockup tampilan menu utama

Gambar 3.14. menampilkan *mockup* dari tampilan menu utama. Terdapat tombol *generate* yang akan memulai proses generasi *terrain* sekaligus memulai *game*, dan tombol *exit* untuk keluar dari *game*. Input box disediakan di pojok kiri bawah layar untuk memasukkan nilai *seed.* Storyboard ditampilkan di tengah layar untuk menampilkan *backstory* dari *game*.

Gambar 3.15. Mockup scene utama

Gambar 3.15. menampilkan mockup dari *scene* utama. Karakter pemain dan *terrain* yang telah dibuat ditampilkan melalui perspektif *top-down*. Tidak ada elemen *user interface* yang ditampilkan dalam *scene* ini.