Algoritmo para la generación procedural de modelos en 3D de zonas urbanas

Mateo Flórez Bacca

Trabajo de Grado para Optar al Título de Ingeniero de Sistemas e Informática

Sergio Augusto Gélvez Cortés

Escuela de Ingeniería de Sistemas e Informática

Master of Science (M.Sc.), Ingeniería de sistemas, Ciencias de la computación

Carlos Jaime Barrios Hernández

Escuela de Ingeniería de Sistemas e Informática

Doctor en Informática y Ciencias Computacionales

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería De Sistemas e Informática

Ingeniería De Sistemas

Bucaramanga

2024

Dedicatoria

Para mi abuela Carmen Rosa, quien fue como una segunda madre y sin necesidad de estudiar filosofía y ética, en su corazón está claro qué es la verdadera bondad y compasión.

Para todas las personas activas del Colectivo por los animales UIS, quienes han sufrido y luchado por aquellos que no tienen voz.

Agradecimientos

Reconozco los aportes del profesor Sergio Augusto y el profesor Carlos Jaime por su orientación y su voluntad de trabajar en un proyecto poco convencional.

Le agradezco a mi madre y padre por su apoyo incondicional durante mi proceso de formación.

Aprecio los esfuerzos de mi novia, por su constante compañía, apoyo y motivación, sin las cuales todos mis proyectos habrían sido mucho más difíciles.

Tabla de Contenido

Pág.

Lista de Tablas

Lista de Figuras

Pág.

Glosario

FPS (Fotogramas por segundo): frecuencia de actualización de la imagen mostrada por la pantalla.

Geometría o Modelado: específicamente en el contexto de este proyecto, se refiere a la existencia dentro de la representación de la escena de volúmenes cuya superficie está coloreada por medio de una textura.

Renderizador: funcionalidad de un software que permite generar una imagen a partir de un conjunto de parámetros.

Renderizar: proceso de generar una imagen haciendo uso de un renderizador.

RNG (Random Number Generator): funcionalidad de un software que permite generar una secuencia de números pseudo-aleatorios.

Shader: programa ejecutado por la GPU que se encarga de realizar cálculos gráficos.

Vista cenital: método para visualizar un lugar en 3D. Consiste en situar la cámara o punto de vista con cierta elevación respecto al suelo y apuntar perpendicularmente al horizonte.

Resumen

Título: Algoritmo para la generación procedural de modelos en 3D de zonas urbanas [1*]

Autor: Mateo Flórez Bacca[2*3*]

Palabras Clave: generación procedural, gráficas por computador, motor de videojuegos.

Descripción: La generación procedural es un conjunto de técnicas que permite hacer uso de computadores para la creación de contenido digital de manera automatizada. Existen múltiples aplicaciones posibles para la generación procedural, una de las cuales tiene como objetivo generar representaciones 2D o 3D de algún tipo de locación. Esta aplicación es ampliamente utilizada en la industria de los videojuegos, dado que algunos proyectos en este medio tiene el requerimiento de contar con escenarios virtualmente infinitos, lo cual es imposible de desarrollar por medio de métodos exclusivamente manuales.

Como parte de una iniciativa de emprendimiento del autor relacionada a la industria de los videojuegos, fue decidido que aprender a diseñar e implementar generación procedural de modelos en 3D de zonas urbanas era necesario. Por lo tanto, este proyecto fue ejecutado como un trabajo de investigación que siguió los pasos requeridos para la creación de un prototipo funcional. Este proceso inició con el estudio y análisis de la literatura sobre la generación procedural. Posteriormente, un diseño original de un algoritmo para la generación procedural de zonas urbanas fue creado. Finalmente, una implementación de este diseño fue desarrollada con un motor de videojuegos de código abierto y ciertas métricas de rendimiento fueron obtenidas y comparadas con varios videojuegos con características relacionadas, tales como compartir el mismo motor y una fidelidad gráfica similar, o contar con generación procedural de escenarios.

[1*] Trabajo de Grado.

[2**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería De Sistemas e Informática. Ingeniería De Sistemas. Director: Sergio Augusto Gélvez Cortés. M.Sc, Escuela de Ingeniería de Sistemas e Informática. Codirector: Carlos Jaime Barrios Hernández. PhD, Escuela de Ingeniería de Sistemas e Informática

[3]

Abstract

Title: Algorithm for the procedural generation of 3D models of urban areas[4]

Author(s): Mateo Flórez Bacca[5]

Key Words: procedural generation, computer graphics, game engine

Description: Procedural generation is a set of techniques that allows taking advantage of computers for the creation of digital content through automatized means. There are many possible applications for procedural generation, one of which aims to generate either 2D or 3D representations of a certain type of location. Such application is widely used in the video-game industry, as some works within this medium have the requirement of featuring virtually infinite scenarios, which is impossible to develop through exclusively authored and hand-crafted methods.

As part of an entrepreneurship initiative from the author related to the video-game industry, it was decided that learning to design and implement procedural generation of 3D models of urban areas was needed. As such, this project was executed as a research work that followed the steps necessary for the creation of a working prototype. These started with the study and analysis of the procedural generation literature. Then, an original design of an algorithm for procedural generation of urban areas was created. Finally, an implementation of this design was developed with an open source game engine and performance results were measured and compared against various video-game works with related features, such as sharing the same engine with a similar graphical fidelity, or having procedural generation of their own.

[4]Graduation Project.

[5]Faculty of Physic-mechanical Engineering, School of Systems and Computing Engineering, Systems Engineering, Director: Sergio Augusto Gélvez Cortés, M.Sc. School of Systems and Computing Engineering. Co-director: Carlos Jaime Barrios Hernández. PhD, School of Systems and Computing Engineering.

Introducción

Uno de los requerimientos de software más comunes en la industria de los videojuegos es la capacidad de visualizar de manera interactiva un escenario. En este contexto, un escenario hace referencia al lugar en donde ocurren los eventos en un videojuego; estos pueden estar representados en 2D o 3D, y tener diferentes temáticas. Para visualizarlos se requiere algún tipo de estructura de datos que permita representarlos. Comúnmente, la manera de producir instancias de dicha estructura de datos consiste en contar con algún software especializado, en el cual un diseñador de niveles define manualmente el escenario deseado. Sin embargo, en algunos proyectos, el rol de diseñador de niveles es reemplazado por un algoritmo, dando lugar a lo que se conoce como *generación procedural de escenarios*.

La generación procedural de escenarios es una técnica importante en la industria, siendo parte de algunos de los proyectos más exitosos de la historia, así como la esencia de varios subgéneros del medio.

Este proyecto tiene como propósito permitirle al autor adquirir conocimientos prácticos respecto al diseño e implementación de algoritmos de generación procedural, por medio de la creación de un software que logre aprovechar exitosamente esta técnica para generar y visualizar interactivamente modelos en 3D de zonas urbanas.

1. Planteamiento y justificación del Problema

Como parte de una iniciativa de emprendimiento del autor, en el futuro se desea desarrollar un videojuego que requiere de un algoritmo para la generación procedural de modelos en 3D de zonas urbanas. Este requisito implica algunos desafíos: el diseño de un algoritmo que realice esta tarea es un problema muy abierto, y su implementación tiene múltiples consideraciones de rendimiento.

Respecto al diseño del algoritmo, es necesario considerar el balance entre características definidas manualmente y aquellas generadas aleatoriamente. Una instancia de la estructura de datos que representa los escenarios cuyos atributos han sido establecidos de manera completamente aleatoria, va a resultar en un escenario sin sentido, donde ningún objeto es reconocible. Contrariamente, una instancia definida manualmente en su totalidad no constituye un producto de generación procedural.

Por otra parte, debido a que la naturaleza de esta investigación está relacionada con algunos aspectos técnicos de la industria de los videojuegos, la implementación de este algoritmo tiene ciertas características de rendimiento que, en el peor de los casos, podrían llegar a considerarse inaceptables (características tales como el tiempo que tarda el programa en completar la generación del modelo 3D, memoria requerida, poder consumido y fluidez de la visualización interactiva). En otras palabras, la optimización de la implementación es sumamente importante.

Por tanto, con la finalidad de adquirir las habilidades necesarias para superar estas dificultades, y siguiendo las indicaciones del Reglamento Estudiantil de Pregrado donde se sugiere que el estudiante desarrolle y fortalezca habilidades propias de sus intereses y proyección profesional al momento de

llevar a cabo su trabajo de grado (Acuerdo 4, Art. 4. 2017), surge la pregunta de investigación: ¿De qué manera se puede diseñar e implementar un algoritmo para la generación procedural de modelos en 3D de zonas urbanas?

2. Objetivos

2.1. Objetivo general

Desarrollar un programa que permita generar proceduralmente modelos en 3D de zonas urbanas y visualizar el resultado de manera interactiva.

2.2. Objetivos específicos

Establecer criterios de aprobación del algoritmo, tales como características del modelo 3D y parametrización requerida.

Establecer criterios de aprobación de la implementación, tales como límites de rendimiento aceptables.

Diseñar un algoritmo para la generación procedural de modelos en 3D de zonas urbanas que se ajuste a los criterios de aprobación previamente definidos.

Implementar en *Godot Engine* el algoritmo junto con una interfaz gráfica para facilitar el proceso de validación.

3. Marco de referencia

3.1. Modelo 3D

En el contexto de gráficas por computador, un modelo 3D es una estructura de datos que representa la geometría de algún objeto en 3D. (Chen, 2009)

3.1.1. Texturas

Una textura es una imagen digital, generalmente en 2D, que se utiliza como fuente para muestras de color de un modelo 3D. Típicamente, a cada vértice del modelo 3D se le asigna una coordenada en la textura. A partir de los vértices es posible interpolar la coordenada de textura de cada punto en la superficie del modelo.

3.1.2. Formato .OBJ

El formato .OBJ representa modelos en 3D por medio de texto plano. Permite definir vértices asignándoles 3 coordenadas y opcionalmente 2 coordenadas de textura, y un vector normal. Posteriormente, se definen superficies planas o líneas que utilizan dichos vértices (FileFormat.Info, s.f.). Para propósitos de este proyecto, no se utilizará la posibilidad de definir vectores normales.

3.2. Motor de videojuegos

Un motor de videojuegos es un kit de desarrollo de software - generalmente integrado bajo una misma interfaz gráfica - que facilita la creación de videojuegos. Sobre esto, Andrade (2015) menciona que un motor de videojuegos adicionalmente permite la reutilización de código y contenido para ser usado en el desarrollo de diferentes juegos.

Los motores aprovechan el hecho de que para la gran mayoría de videojuegos hay requerimientos muy similares, tales como:

- ✓ Renderizador para gráficas en 2D y/o 3D.
- ✓ Lectura de distintos dispositivos de entrada, como teclados y controles.
- ✓ Una estructura de datos para la representación de escenas, así como un editor especializado para generar instancias de esta estructura.
- ✓ Administrador de recursos en memoria.
- ✓ Soporte para reproducir distintos formatos de audio.
- ✓ Un modelo para la creación de entidades y eventos de juego, junto con un mecanismo para iterar dicho modelo.

En particular, este proyecto requiere de un renderizador para gráficas en 3D, una estructura de datos para la representación de escenas, y acceso al estado del teclado y el mouse.

3.2.1. Godot Engine

Godot Engine es un motor de videojuegos gratuito y de código abierto, publicado bajo la licencia MIT, y originalmente desarrollado por Juan Linietsky y Ariel Manzur. (Godot, s.f)

Algunos aspectos importantes de este motor a tener en cuenta son:

Modelo híbrido. La representación de escenas y entidades del juego se maneja bajo un mismo mecanismo: nodos. Una escena se define como una jerarquía de nodos, donde cada nodo es una instancia de una clase particular. Debido a esto, Godot utiliza herencia y composición simultáneamente para representar sus escenas. (Linietsky, 2019)

Contenido 3D. Godot carece de un editor de modelos en 3D, por lo que es necesario importar la geometría a partir de un archivo externo. Para efectos de este proyecto, se utilizará el formato .OBJ junto con el nodo MeshInstance. Esta clase de nodo permite renderizar instancias de un modelo 3D. (Godot Docs, s.f)

Nodos y recursos. Para la creación de software en Godot es importante familiarizarse con dos clases: nodos y recursos (*Node* y *Resource*). *Resource* es la clase base para todos los tipos de recursos específicos de Godot, funcionando principalmente como contenedores de datos (documentación oficial de Godot). *Node* es la clase base con la que se definen las escenas. Las instancias de esta clase o de sus derivados se pueden configurar en una estructura de árbol (es decir, cada nodo tiene un pariente y puede tener múltiples hijos). Este árbol es la forma en como se representa una escena (Godot Docs, s.f).

Para este proyecto los recursos más relevantes son *Texture* y *Mesh. Texture* permite registrar una imagen en la GPU. *Mesh* almacena geometría basada en arrays de vértices (Godot Docs, s.f).

3.3. Generador de números pseudo aleatorios (RNG)

Un generador de números pseudo aleatorios es un algoritmo determinístico capaz de generar una secuencia de números que aparentemente siguen cierta distribución de probabilidad. (Johnston, 2018)

La importancia de estos algoritmos yace en que para muchas aplicaciones, es necesario realizar cálculos que requieren de números cuyo comportamiento es aleatorio. Sin embargo, los RNG son procesos deterministas, por tanto requieren de un valor de inicialización al que se le llama “semilla” si se desea que la secuencia de números generados sea distinta cada vez que se realiza el proceso.

La gran mayoría de lenguajes de programación utilizados en el desarrollo de videojuegos proveen algún mecanismo simple para adquirir una secuencia de números a partir de un RNG.

3.4. Generación procedural de contenido (PCG)

PCG es el conjunto de técnicas que permiten automatizar la producción de contenido. En este contexto, “contenido” hace referencia a aquellos trabajos que tradicionalmente requieren de un autor humano: música, poesía, dibujos arquitectónicos, modelados 3D, etcétera. La PCG abarca distintos

paradigmas y estrategias. A continuación se hará referencia a algunos de los conceptos relacionados con la PCG que potencialmente representan un beneficio para los objetivos de este proyecto.

3.4.1. Generación procedural de escenarios

La generación procedural de escenarios es una de las categorías de la PCG más importantes en la industria de los videojuegos; este es el enfoque principal de este proyecto. Consiste en algoritmos para la automatización de la producción de instancias de alguna estructura de datos que represente un escenario. Este tipo de algoritmo es altamente dependiente de la estructura de datos utilizada, así como de las características deseadas en los escenarios generados. En otras palabras, gran parte del diseño de un algoritmo de este estilo es un proceso creativo, no técnico. No obstante, existen algunos conceptos recurrentes a la hora de diseñar un generador procedural de escenarios. Algunos de estos son:

Restricciones. En algunos casos, es deseable que ciertos patrones nunca aparezcan en las escenas (por ejemplo, el diseñador de algoritmo quizás quiera definir que sea imposible generar un cementerio junto a una discoteca). Una vez identificados estos patrones, es posible programar excepciones para evitar que ocurran. (Short. Adams. 2017)

Branching: Consiste en un proceso generativo en el cual se definen porciones del escenario aledañas a aquellas porciones que ya fueron definidas. Usualmente esto implica que los extremos del escenario son callejones sin salida. (Short. Adams. 2017)

Generación cíclica. Un método opuesto al branching, en el cual se intenta que exista más de un camino desde dos ubicaciones arbitrarias en el escenario. Se puede implementar de diversas formas, pero un método robusto consiste en definir un grafo de ubicaciones deseadas y sus conexiones antes de definir el escenario. (Short. Adams. 2017)

Método paramétrico: el método paramétrico es una técnica ampliamente utilizable en distintos tipos de PCG. En general, consiste en definir manualmente algún contenido deseado como

una colección de parámetros. Posteriormente, se genera una permutación aleatoria de dichos parámetros. La ventaja de este método consiste en que la cantidad de posibilidades está en función de la combinatoria entre parámetros. (Short. Adams. 2017)

3.5. Antecedentes en la generación procedural de escenarios

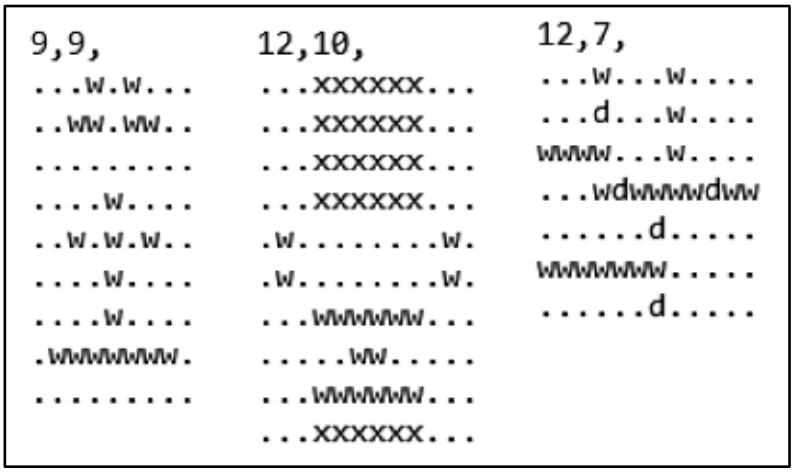
A continuación se incluye una lista de algunas de las técnicas específicas utilizadas en algoritmos de generación procedural de escenarios.

3.5.1. Generación de calabozos en Dungeonmans

Dungeonmans es un videojuego cuyos escenarios utilizan una cuadrícula como estructura de datos para la representación de escenas en 2D. A cada cuadrícula le corresponde un estado que indica el tipo de objeto o terreno en esa localización. Esta representación se conoce como *Tilemap*. (Godot Docs, s.f.) Su algoritmo de generación procedural de escenarios está diseñado para generar calabozos. En términos generales, funciona de la siguiente esta manera:

Primero, se debe contar con cierto contenido definido manualmente: porciones pequeñas de escenario que se almacenan como texto plano. Estas porciones podrían considerarse como las habitaciones con las que se va a generar el calabozo.

Figura 1. *Habitaciones predefinidas en Dungeonmans.*



Nota: Short, T., & Adams, T. (2017). Procedural Generation in Game Design (1st ed.). A K Peters/CRC Press. DOI: 10.1201/9781315156378.

En la figura 1 (Short. Adams. 2017) se muestran algunos ejemplos de habitaciones definidas manualmente. Se interpretan de esta forma:

Primera línea: dimensiones de la habitación

x: espacio no usado

w: pared

. (punto): piso

d: puerta

Una vez se tienen las habitaciones deseadas, el algoritmo utiliza generación con branching de la siguiente manera:

Se definen las dimensiones del calabozo y se ubica una habitación aleatoria en cualquier lugar. El algoritmo sigue un proceso iterativo en el que añade nuevas habitaciones en el perímetro de las porciones de escenario que ya fueron definidas.

Dicho proceso iterativo sigue el orden mencionado a continuación.

a) Generar una lista de todas las paredes que cumplan con las siguientes condiciones:

- Tienen dos paredes aledañas en el mismo eje.
- Tienen una celda aledaña de espacio sin utilizar.

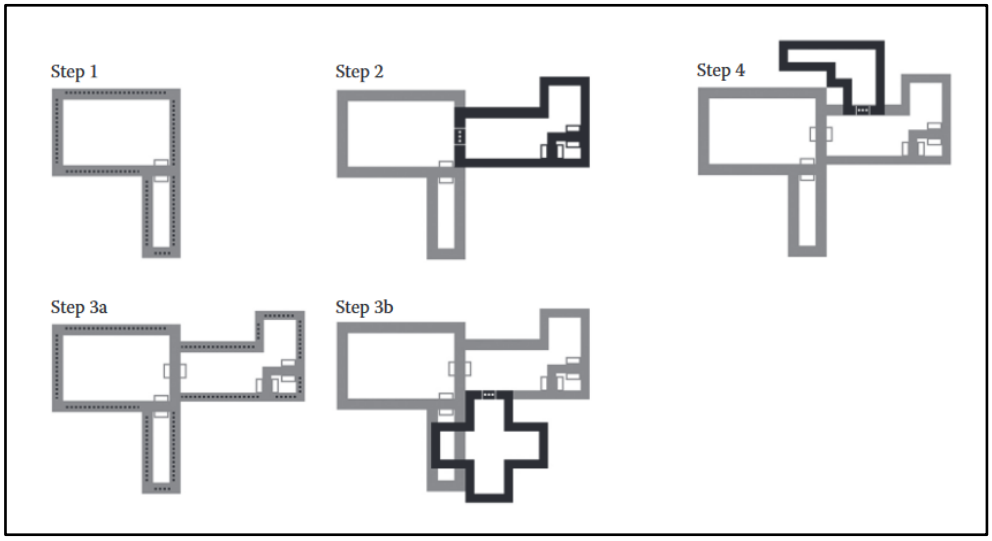
Esta va a considerarse la lista de paredes que son *candidatas.*

b) Seleccionar aleatoriamente una de las candidatas. Adicionalmente, seleccionar aleatoriamente una de las habitaciones predefinidas y ubicarla en contacto con la celda candidata, en cualquier alineamiento.

c) Si la habitación encaja, se reemplaza la celda candidata por una puerta y se recalcula la lista de candidatas.

- Si la habitación no encaja, se regresa al paso 1 y se intenta el proceso nuevamente con otra candidata.
- Una vez ubicada la nueva habitación, se revisa si existen celdas vacías adyacentes a una celda tipo piso, y se reemplazan por celdas tipo pared.

Figura 2. *Diagrama de las etapas del proceso iterativo en Dungeonmans.*



Nota: Short, T., & Adams, T. (2017). Procedural Generation in Game Design (1st ed.). A K Peters/CRC Press. DOI: 10.1201/9781315156378.

Este proceso representado en la figura 2 (Short. Adams. 2017), continúa hasta que el espacio disponible no sea suficiente para una nueva habitación, o hasta que se alcance una cantidad deseada de habitaciones.

3.5.2. Parámetros globales en Sid Meier's Civilization

Sid Meier's Civilization es un videojuego cuyos escenarios son mapamundis similares al de la tierra, donde cada partida es diferente gracias a un algoritmo de generación procedural. Cada vez que el algoritmo genera un nuevo mundo, utiliza una serie de parámetros globales que afectan el tipo de terreno generado:

Masa terrestre. Define la proporción entre océanos y tierra. Un valor bajo genera más islas. Un valor alto genera más continentes.

Temperatura. Controla la proporción de biomas áridos a lo largo del ecuador. Un valor bajo genera más tundras. Un valor alto genera más desiertos.

Clima. Controla la proporción de biomas húmedos. Un valor bajo genera más desiertos. Un valor alto genera más junglas y pantanos.

Era temporal. Controla la erosión del terreno. Un valor bajo genera más montañas. Un valor alto genera más cerros y lagos.

3.5.3. Método paramétrico para contenido predefinido en Spelunky

Spelunky es un videojuego de plataformas en 2D que utiliza tilemaps para la representación de sus escenarios. Su algoritmo de generación procedural está diseñado para producir cuevas que contienen ciertos objetos especiales, tales como urnas, bloques de piedra, escaleras y tesoros. Cada escenario consiste de 16 pantallas en una cuadrícula de 4x4. A cada pantalla se le asigna aleatoriamente una porción de escenario definido manualmente; sin embargo, cada una de estas porciones se ve modificada dinámicamente, añadiendo objetos en lugares aleatorios. La configuración de pantallas distribuidas aleatoriamente y modificadas dinámicamente permite generar un gran número de permutaciones de escenarios posibles.

Figura 3. *Resultado del generador procedural de escenarios de Spelunky.*

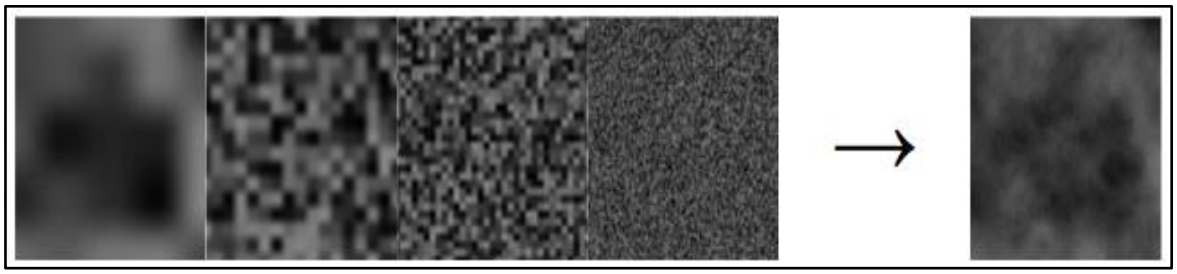
3.5.4. Perlin noise para la generación de heightmaps en Minecraft

Minecraft es un videojuego en 3D cuyos escenarios son principalmente biomas naturales que se generan proceduralmente. Para ello, una de las técnicas que aprovecha Minecraft es el uso de *heightmaps*. Un heightmap es una cuadrícula en 2D donde cada casilla contiene un valor numérico y los valores de las casillas varían suavemente a lo largo del espacio (Short. Adams. 2017). Los heightmaps tienen múltiples aplicaciones. En el caso de la generación procedural de escenarios en Minecraft, la elevación del terreno se define usando un heightmap. Este heightmap se calcula utilizando Perlin noise, un algoritmo que permite generar valores aleatorios en cuadrículas de N-dimensiones, donde para cualquier celda, los valores de su vecindario mantienen coherencia local. (Short. Adams. 2017)

El funcionamiento del algoritmo requiere generar una secuencia de números a partir de un RNG. Esta secuencia se considera la “semilla”, y debe ser mapeada a un espacio de N-dimensiones dependiendo de la aplicación. Posteriormente se realiza una interpolación de valores para las posiciones entre los puntos de la semilla. Se repite este proceso para X espacios de mayor frecuencia. Finalmente,

se suma el resultado de todas las frecuencias, típicamente decreciendo la amplitud de las frecuencias más altas. Ver figura 4 (Kelly. McCabe. 2017)

Figura 4: *Visualización de espacios de distintas frecuencias y el resultado al sumarlos.*



Nota: Kelly, G. McCabe, H. (2006) A Survey of Procedural Techniques for City Generation, The ITB Journal: Vol. 7: Iss. 2, Article 5. DOI:10.21427/D76M9P.

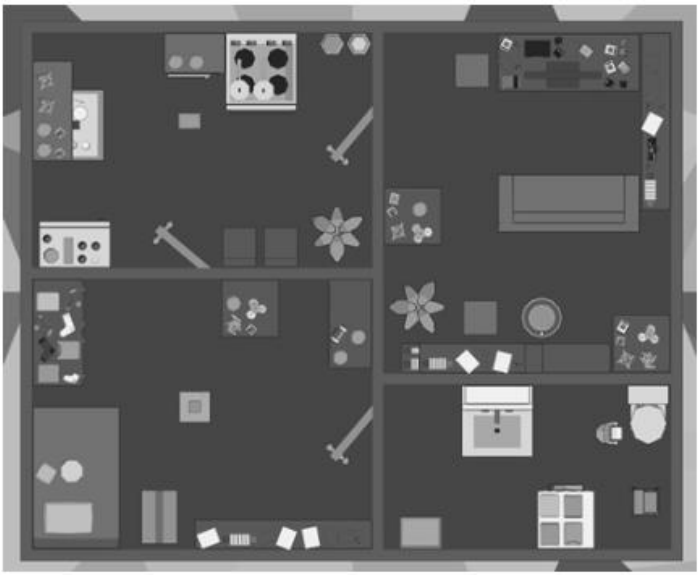
Figura 5. *Resultado de la generación procedural en Minecraft*

3.5.5. Generación de planos en 2D para escenarios 3D en Catlateral Damage

Catlateral Damage es un videojuego en 3D cuyos escenarios representan los espacios al interior de una casa, y cada una de estas zonas se genera proceduralmente. Debido a que para propósitos del

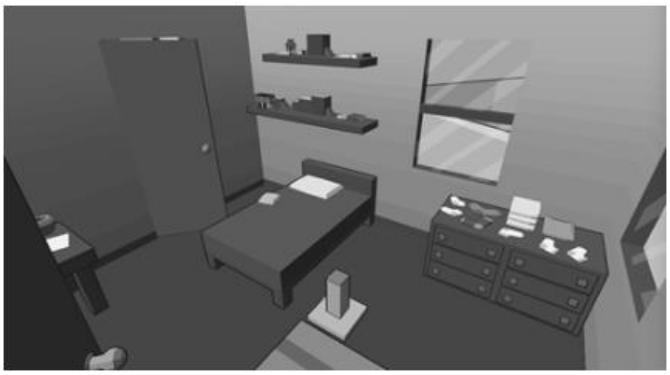
juego no es necesario generar casas con varios pisos, su algoritmo de generación procedural de escenas aprovecha la simplicidad del 2D para crear una representación intermedia del producto final. Este algoritmo funciona en 3 etapas: primero, se define una región rectangular grande, la cual es "cortada" en múltiples secciones que representan las distintas habitaciones. Luego, se asigna un tipo de habitación a cada sección y se ubican los objetos relevantes en cada una, tales como muebles y decoraciones. Finalmente, se aprovechan estos planos para generar una versión 3D de la escena. Ver figuras 6 y 7 (Short. Adams. 2017).

Figura 6. *Representación intermedia en Catlateral Damage.*



Nota: Short, T., & Adams, T. (2017). Procedural Generation in Game Design (1st ed.). A K Peters/CRC Press. DOI: 10.1201/9781315156378.

Figura 7. *Resultado final de la generación procedural en Catlateral Damage.*



Nota: Short, T., & Adams, T. (2017). Procedural Generation in Game Design (1st ed.). A K Peters/CRC Press. DOI: 10.1201/9781315156378.

4. Metodología

4.1. Primera etapa: definición de requerimientos

4.1.1. Establecer criterios de aprobación del algoritmo, tales como características del modelo 3D y parametrización requerida

Los detalles internos del funcionamiento del algoritmo se consideran irrelevantes para su aprobación. En su lugar, son de interés sus datos de entrada y el resultado que este produzca. Ya que el algoritmo debe encargarse de generar geometría, los criterios de aprobación de este deberían estar ligados a una especificación de geometría deseada. Existen virtualmente infinitas especificaciones posibles, así que no existe una respuesta directa a la hora de definir una especificación. Es necesario considerar de qué manera se van a aprobar o descartar características deseadas. En este caso, se decidió tener en cuenta dos factores: El primer factor consiste en considerar qué características son deseables

y qué características son irrelevantes en el contexto del emprendimiento planeado del autor. El segundo factor consiste en considerar qué características deben descartarse debido a que tienen como requisito lidiar con una complejidad que se sale del alcance del proyecto. Este segundo factor se basa en los conocimientos adquiridos al estudiar la literatura respecto a generación procedural de escenarios. Por otro lado, la especificación de la geometría simplemente se encarga de establecer una serie de características geométricas, lo cual se puede satisfacer con diversas elecciones de parámetros globales. La parametrización está entonces fuertemente ligada al diseño del algoritmo más no a la especificación de la geometría. Por tanto, una vez decidido el funcionamiento general del algoritmo, se deben identificar las posibilidades de parametrización para aquellos valores que afectan globalmente a la generación y que originalmente eran constantes.

4.1.2. Especificación del tipo de geometría deseada

A partir del proceso previamente descrito, se define una lista de características, junto con su justificación:

Variedad en el tipo de lugares (hospitales, parques, etc.). Desde la perspectiva del diseño de videojuegos, este requerimiento generalmente es crucial.

Carencia de realismo arquitectónico o consideraciones de ingeniería civil. Nuevamente desde una perspectiva de diseño de videojuegos, típicamente se priorizan otros aspectos sobre el realismo.

Geometría de baja fidelidad. Incrementar la fidelidad en este caso simplemente es un costo adicional sin ningún beneficio real respecto al estudio de la generación procedural.

Carencia de variación en el nivel del suelo. Es un compromiso que se acepta debido a la complejidad de considerar geometría con distintas pendientes y orientaciones mientras que se garantice que la superficie resultante sea continua.

Carencia de realismo en el sistema de vías. Basándose en un estudio (A Survey of Procedural Techniques for City Generation), el autor identificó que la complejidad de un generador procedural de un sistema de vías es mayor a la complejidad de un generador procedural de ciudades cuyas vías son una cuadrícula perfecta, por lo tanto este requerimiento se sale de las posibilidades del proyecto.

Carencia de geometría al interior de las edificaciones. Nuevamente es un compromiso que se toma debido a la complejidad adicional que implica. El modelado al interior de las edificaciones, desde un punto de vista geométrico, puede tener una complejidad similar a aquel de los exteriores. Adicionalmente, la gran mayoría de edificaciones no son accesibles en los escenarios de videojuegos con zonas urbanas amplias.

4.1.3. Establecer criterios de aprobación de la implementación, tales como límites de rendimiento aceptables

La motivación detrás del establecimiento de límites de rendimiento aceptables tiene que ver con la habilidad para desarrollar productos que cumplan con los estándares de la industria. Por tanto, se decidió realizar un análisis del perfil de rendimiento de distintos productos comerciales exitosos que cuenten con algunas de las mismas funciones que se desean implementar, tales como generación procedural de escenarios y visualización interactiva de escenarios en 3D. A partir de los resultados de este análisis se establece si la implementación es aceptable. Se define entonces que la implementación es exitosa si logra estar por encima de la media en las métricas de rendimiento.

Existen varias métricas que permiten medir la optimización de un videojuego. Algunas de estas si bien son importantes para un proyecto comercial se consideraron de bajo riesgo debido a la

naturaleza de esta implementación. Por ejemplo, el tamaño en disco de la implementación es de poco interés, debido a que los archivos fuente del proyecto consisten de algunos scripts e imágenes. El resto del tamaño en disco proviene del runtime de Godot, el cual es naturalmente aceptable para productos comerciales en la industria. Adicionalmente, la memoria utilizada por lo general únicamente llega a ser un problema en videojuegos con modelado y contenido de alta fidelidad. Por el contrario, la visualización en tiempo real y el tiempo de generación podrían tener un rendimiento inaceptable si la implementación tiene suficientes defectos. Es por esto que se diseñaron dos pruebas para validar estas métricas, las cuales se definen a continuación:

Tabla 1. Pruebas para la validación del rendimiento

Prueba	Descripción	Método	Criterio de aprobación
FPS	Fotogramas por segundo. Un valor alto es mejor.	Se ejecuta el programa sin ninguna otra aplicación simultáneamente. Con un dispositivo externo, se captura un video del historial de FPS por 20 segundos y se calcula el promedio.	La implementación está por encima de la media.
Tiempo de generación.	Definido en segundos: tiempo que le toma al programa generar un escenario. Un valor bajo es mejor.	Se ejecuta el programa sin ninguna otra aplicación simultáneamente. Con un dispositivo externo, se captura un video del proceso de generación. Con un software de edición de video se hace un conteo exacto de fotogramas.	La implementación está por debajo de la media.

4.2. Segunda etapa: diseño

4.2.1. Diseñar un algoritmo para la generación procedural de modelos en 3D de zonas urbanas que se ajuste a los criterios de aprobación previamente definidos.

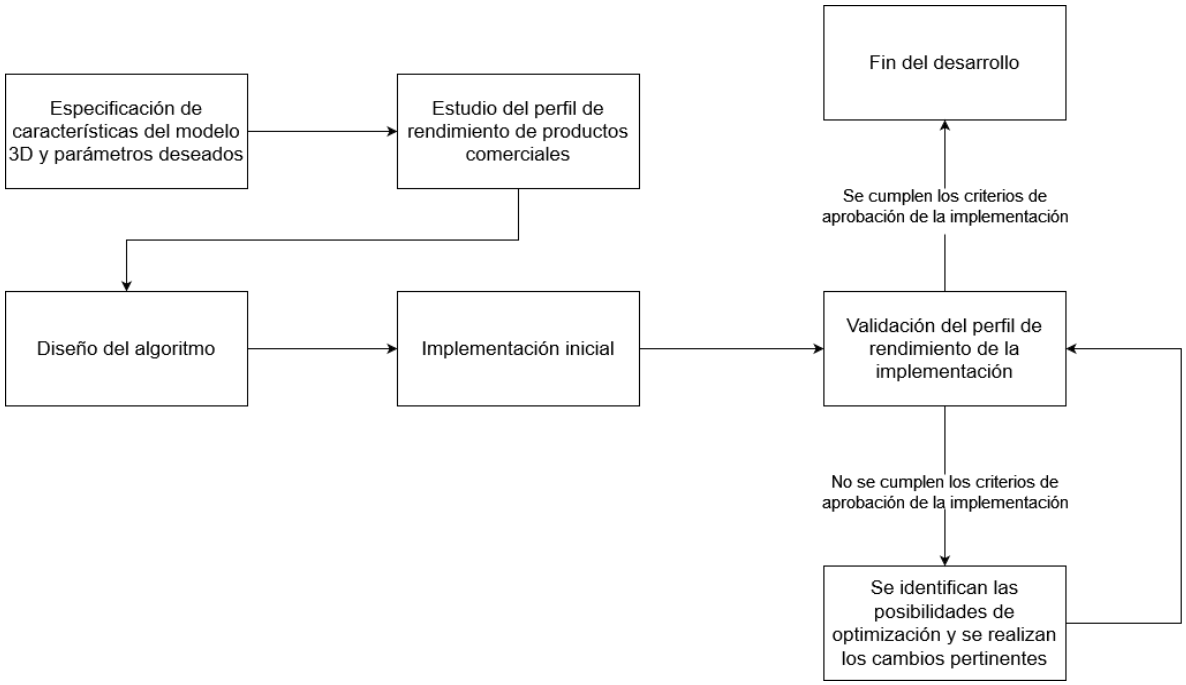
Esta etapa del proyecto es donde la gran mayoría de la investigación rinde sus resultados. Se profundizó en la literatura y se aplicaron las distintas técnicas de generación procedural aprendidas junto con una exploración de métodos originales. El resultado de esta etapa naturalmente es un algoritmo, el cual se especifica a profundidad con un diagrama de flujo junto con varias secciones de este documento.

4.3. Tercera y cuarta etapa: implementación, validación y optimización

4.3.1. Implementar en Godot Engine el algoritmo junto con una interfaz gráfica para facilitar el proceso de validación.

Respecto a estas últimas etapas del proyecto, hay 2 conceptos claves a tener en cuenta: optimización y validación. Se profundizó en la documentación de Godot con el fin de identificar las posibilidades de optimización del rendimiento. Posteriormente, se incluyó una interfaz gráfica que permite establecer los parámetros de entrada del algoritmo, así como mostrar dinámicamente las métricas de rendimiento en tiempo real, de tal manera que la captura de datos para el proceso de validación sea sencilla.

Figura 8. *Diagrama de flujo de las actividades a realizar.*



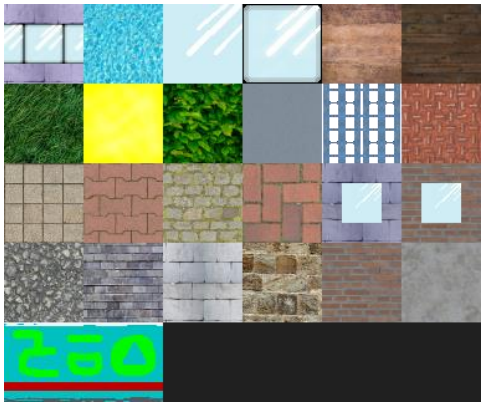
5. Aspectos generales de diseño e implementación del algoritmo

5.1. Materiales globales

Los nodos *MeshInstance* necesitan una referencia a un recurso de tipo *Material.* Para esta implementación se utilizaron recursos *SpatialMaterial*, una clase derivada a *Material.* Un *Material* es un recurso abstracto que permite colorear y sombrear geometría, mientras que un *SpatialMaterial* es una implementación de este recurso, la cual provee una amplia variedad de opciones y propiedades de renderizado sin necesidad de escribir un shader (documentación oficial de Godot). De todas estas opciones, únicamente se utilizó una textura para definir el color.

Ya que existe una única escena en 3D, todas las texturas se utilizan simultáneamente. Por lo tanto no es necesario administrar dinámicamente estos recursos en memoria. Es decir, cada *SpatialMaterial* se crea al inicio del programa y se define una variable global con una referencia a este.

Figura 9. *Texturas utilizadas en la geometría 3D.*



5.2. Arrays de 2 dimensiones

Una de las estructuras de datos más utilizadas en este proyecto es el array de 2 dimensiones. Consiste en una cuadrícula donde cada celda almacena un valor. El acceso a cada celda se realiza por medio de dos índices, uno para cada dimensión. Existen múltiples maneras de implementar esta estructura. En este caso, se decidió utilizar internamente un array de una dimensión cuyo tamaño es de alto por ancho (cantidad de celdas por dimensión). El acceso a cada celda se calcula por medio de la fórmula X + WY, donde X es el índice en la primera dimensión, Y es el índice en la segunda dimensión y W es la cantidad de celdas en la primera dimensión.

5.3. Cuadrícula de chunks

Una de las optimizaciones más importantes en el campo de las gráficas por computador en tiempo real consiste en procesar únicamente aquellos elementos de la escena que son visibles desde la perspectiva actual de la cámara. Considerar esto es importante para este proyecto, ya que las ciudades generadas contienen decenas de manzanas y procesarlas todas al mismo tiempo sería ineficiente. La solución utilizada en este proyecto divide la totalidad de la geometría en una cuadrícula (vista desde el plano cenital) de celdas de tamaño uniforme, en donde a cada celda se le asocia una lista con los nodos de Godot que representan la geometría dentro de dicha celda. Una vez establecida esta cuadrícula, es

posible calcular en cuál celda se ubica actualmente la cámara (celda central). Tomando la celda central como origen se define un área de 5x5 celdas. Cada vez que hay un cambio en la posición de la celda central, se redefine esta área, se desactivan todos los nodos que no pertenezcan a esta y se activan todos los nodos dentro de ella.

5.4. Generación de primitivos

Un primitivo es un elemento geométrico simple, y una escena se compone de un conjunto de primitivos (Peter Shirley, 2016).

Para cualquier algoritmo de generación procedural de escenarios es necesario contar con funcionalidad que permita añadir elementos geométricos. La manera específica en cómo se define esta funcionalidad depende del tipo de renderizador de gráficos y del tipo de escenarios deseados. Debido a que Godot es un motor para la creación de videojuegos, su renderizador de gráficos en 3D utiliza prácticas y estructuras de datos que son estándares en la industria. Por ejemplo, la geometría en 3D se define como un conjunto de triángulos que cubren la superficie de los objetos, donde cada vértice define una posición en 3D, una coordenada de textura en 2D y un vector normal de 3 componentes. Es decir, para este proyecto es necesario que la funcionalidad que añade geometría a la escena se adhiera a este formato. Sin embargo, definir cada triángulo manualmente sería excesivamente impráctico, por lo que se crearon 3 funciones que se consideran los primitivos del algoritmo: secciones rectangulares de planos horizontales alineadas a los ejes, cuboides alineados a los ejes y secciones rectangulares de planos verticales alineadas al eje vertical. Cada una de las funciones que genera estos primitivos requiere las coordenadas y dimensiones específicas, de la celda en la cuadrícula de chunks y de la textura utilizada.

5.5. Implementación en Godot de las funciones generadoras de primitivos

Por medio de la clase *ArrayMesh*, Godot permite la creación de un recurso *Mesh* cuyos atributos son establecidos desde un conjunto de arrays (Godot Docs, s.f). Una de las posibles aplicaciones de esta clase es la creación de modelos en 3D por medio de código fuente. Para esta implementación, la forma como se generan los primitivos del algoritmo previamente descritos consiste en definir una serie de funciones que toman como parámetros las dimensiones, posición y características del primitivo (tales como el material que utiliza y el chunk al que pertenece). A partir de estos parámetros, se genera dinámicamente un conjunto de arrays que contienen los valores necesarios para describir los vértices, coordenadas de textura y vectores normales de cada triángulo de la superficie del primitivo. Una vez realizado este proceso se cuenta con el recurso *Mesh* adecuado, pero aún es necesario instanciar un nodo *MeshInstance.* Este nodo es el más utilizado para visualizar instancias de un *Mesh* en la escena (Godot Docs, s.f). Finalmente se añade una referencia a este nodo a la cuadrícula de chunks y se inserta el nodo en la escena.

5.6. Cámara dinámica

Para visualizar una escena de manera dinámica en Godot (es decir, cambiando la posición y ángulo de vista a lo largo del tiempo) es necesario hacer uso del nodo *Camera*, el cual se encarga de mostrar en pantalla la escena desde su posición actual. Enlazando este nodo junto con un script, es posible especificar una serie de instrucciones que cambien la posición y el ángulo en función del estado del teclado y el mouse. En este caso, las teclas de W, A, S y D permiten mover la cámara por el plano horizontal; control y barra espaciadora por el plano vertical y el movimiento horizontal y vertical del mouse ajustar la guiñada y la inclinación.

5.7. Parámetros globales

El algoritmo cuenta con algunos parámetros globales:

Probabilidad de bloque especial. Probabilidad para cada manzana de ser considerada un bloque especial. Los bloques no especiales simplemente contienen casas y edificios, mientras que los especiales son otro tipo de lugares (hospitales, institutos, etc.).

Tamaño del mundo. Definido en cantidad de manzanas por dimensión.

Densidad de edificios en el centro. La densidad de edificios para bloques no especiales es la probabilidad de que se genere un edificio en lugar de una casa. Este parámetro define la densidad de edificios para la celda central.

Caída en la densidad de edificios. Junto con la densidad central, permite calcular la densidad de edificios para cualquier celda en función de la distancia con la celda central.

Semilla para el RNG. Define el estado inicial del generador de números pseudo aleatorios. En este caso la semilla es un campo de texto plano al cual se le aplica una función hash.

Adicionalmente, hay un parámetro gráfico opcional que permite activar la funcionalidad de sombreado automático de Godot.

5.8. Estructura general de la ciudad

Existen múltiples tipos de manzanas posibles. Para cada uno de estos tipos, se define un procedimiento que se encarga de generar la manzana. Estos procedimientos fueron llamados “generadores”. Internamente, los generadores quizás necesiten otro procedimiento para generar una edificación con elementos aleatorios pero que no constituye una manzana (como una casa, un edificio o una estatua). Para estos sub-procedimientos se utilizó el nombre de “sub-generadores”. Adicionalmente, también hay ciertos elementos geométricos que aparecen en múltiples generadores pero que carecen de aleatoriedad (como bancas, postes de luz o árboles). Estos también cuentan con

un procedimiento y fueron denominados “props”. A continuación se presenta una lista de cada uno de los procedimientos:

a) Generadores

1. Manzana normal (cuenta con casas y edificios)
2. Hospital
3. Instituto
4. Parque
5. Conjunto residencial

b) Sub-generadores

1. Edificio
2. Cafetería
3. Casa
4. Estatua
5. Andén

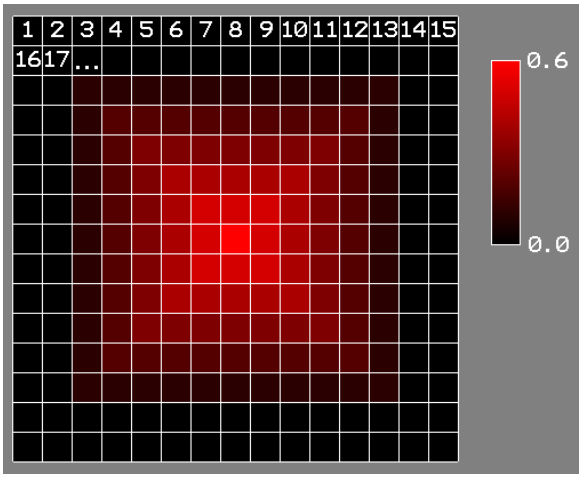
c) Props

1. Banca
2. Poste de luz
3. Árbol

La ciudad consiste en una cuadrícula perfecta de manzanas. La cuadrícula se recorre iterativamente y para cada casilla se genera una manzana. El generador utilizado para cada manzana va a depender del parámetro de probabilidad de bloque especial y de una llamada extra al RNG para definir el tipo de bloque especial en caso que sea necesario.

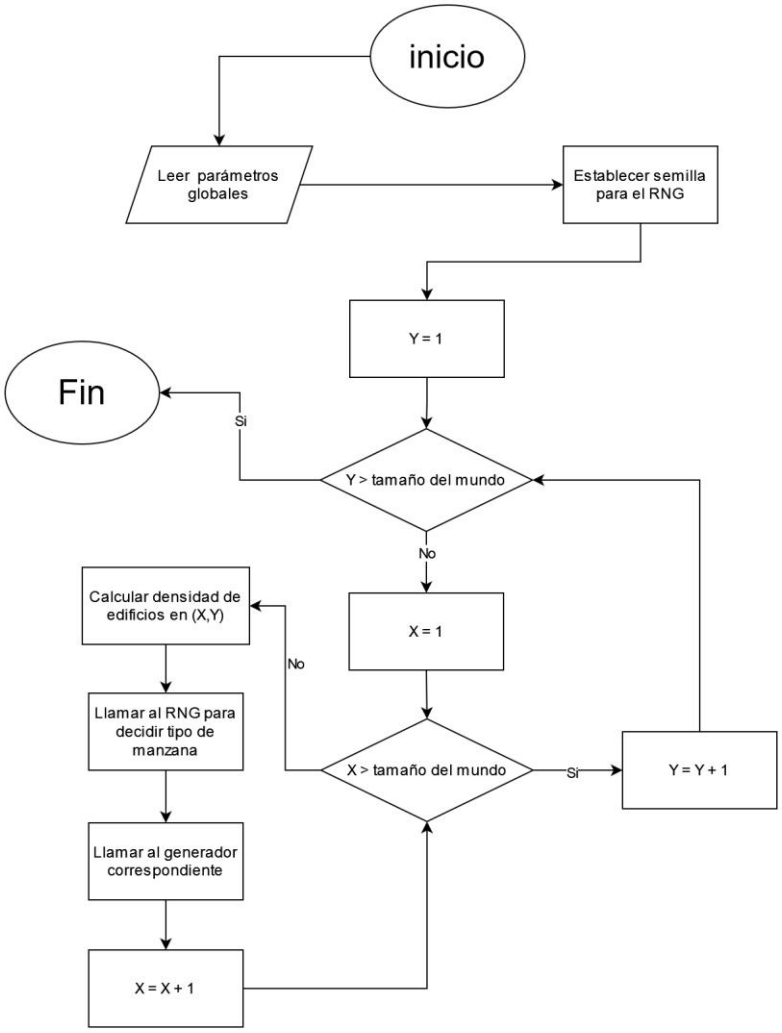
El generador de manzanas normales cuenta con un parámetro único, el cual se interpreta como la densidad de edificios (entre 0 y 1). Este valor es la probabilidad de cada casa de ser reemplazada por un edificio. La densidad está más concentrada en las manzanas centrales.

Figura 10. *Diagrama de distribución de densidad de edificios*



Nota: Para los parámetros por defecto se muestra la distribución de la densidad de edificios. Adicionalmente se muestra el orden de generación de las manzanas.

Figura 11. *Diagrama de flujo para la generación de la ciudad*

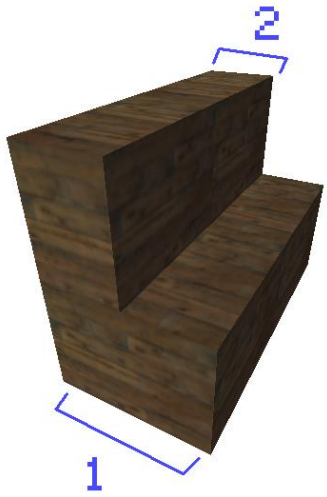


6. Contenido específico del diseño

6.1. Prop: banca

La función para generar bancas toma como parámetros la coordenada central donde será creada (desde la vista cenital), la elevación respecto al suelo y el ángulo hacia el cual apunta el asiento. Se construye por medio de dos cuboides de tamaño fijo (señalados con 1 y 2 en la imagen). Adicionalmente, se define un nodo tipo *Spatial* (nodo base para 3D) ubicado en las coordenadas centrales previamente mencionadas y se establece como el pariente de los cuboides. Al aplicarle rotación al nodo pariente esta se propaga por los nodos descendientes, permitiendo así rotar el objeto completo.

Figura 12. *Apariencia in-engine de una banca.*

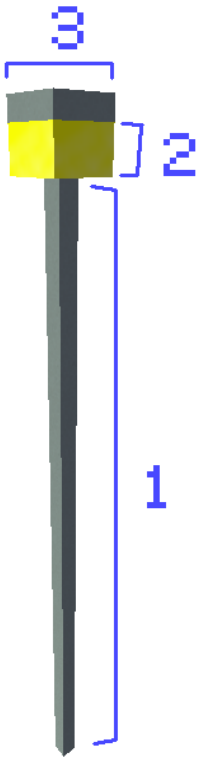


Nota: Se señalan los cuboides que la conforman

6.2. Prop: poste de luz

La función para generar postes de luz toma como parámetros la coordenada en el plano horizontal y la elevación respecto al suelo. Se construye por medio de tres cuboides de tamaño fijo (señalados con 1, 2 y 3 en la imagen).

Figura 13. *Apariencia in-engine de un poste de luz.*

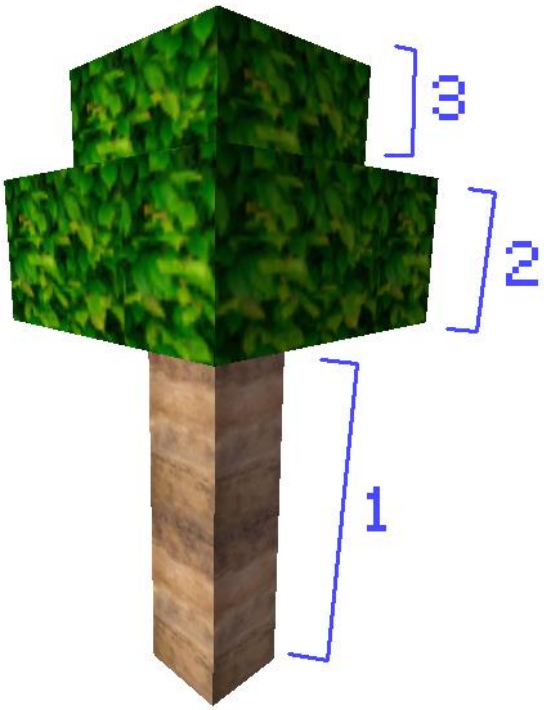


Nota: Se señalan los cuboides que la conforman.

6.3. Prop: árbol

La función para generar árboles toma como parámetros la coordenada en el plano horizontal, la elevación respecto al suelo y un escalar que multiplica algunos parámetros de los primitivos, de tal manera que se puedan generar árboles de distintos tamaños. Se construye por medio de tres cuboides (señalados con 1, 2 y 3 en la imagen).

Figura 14. *Apariencia in-engine de un árbol.*



Nota: Se señalan los cuboides que lo conforman. Para este caso se utilizó escala = 1.0.

6.4. Sub-generador: edificio

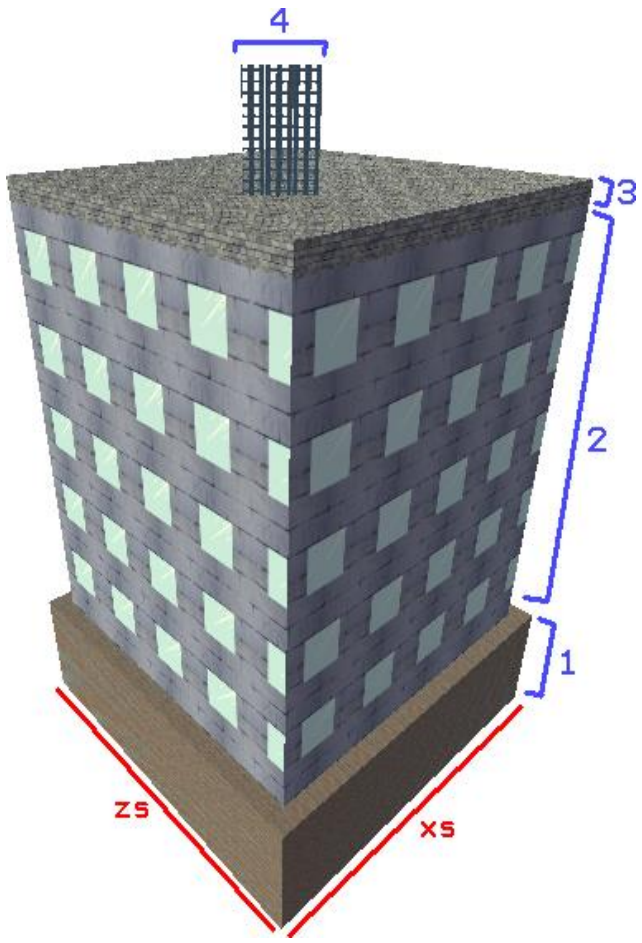
El sub-generador de edificios toma como parámetros la elevación respecto al suelo, una coordenada en el plano horizontal, las dimensiones de la base (*xs* y *zs* en la imagen) y un número entero como semilla. Esta coordenada define la esquina noroeste. Cada edificio se construye por medio de tres cuboides (1, 2 y 3 en la imagen). El cuboide 2 es generado con una textura escogida aleatoriamente entre un conjunto de texturas posibles. Los cuboides 1 y 3 son de tamaño fijo, mientras que el cuboide 2 depende de una fórmula:

$$h = randirange(3,6) * H$$

La función randirange (a,b) retorna un número entero en el rango [a, b] con distribución uniforme. H es una constante. Esta fórmula debe interpretarse como escoger aleatoriamente entre una altura de 3 a 6 pisos.

Finalmente, el sub-generador tiene una probabilidad de 0.3 de generar una antena en el tejado (4 en la imagen). Para este caso no se utilizó un prop, sino dos llamadas al primitivo de secciones rectangulares verticales, cuyas coordenadas hacen que estén cruzadas en X.

Figura 15. *Apariencia in-engine de un edificio.*



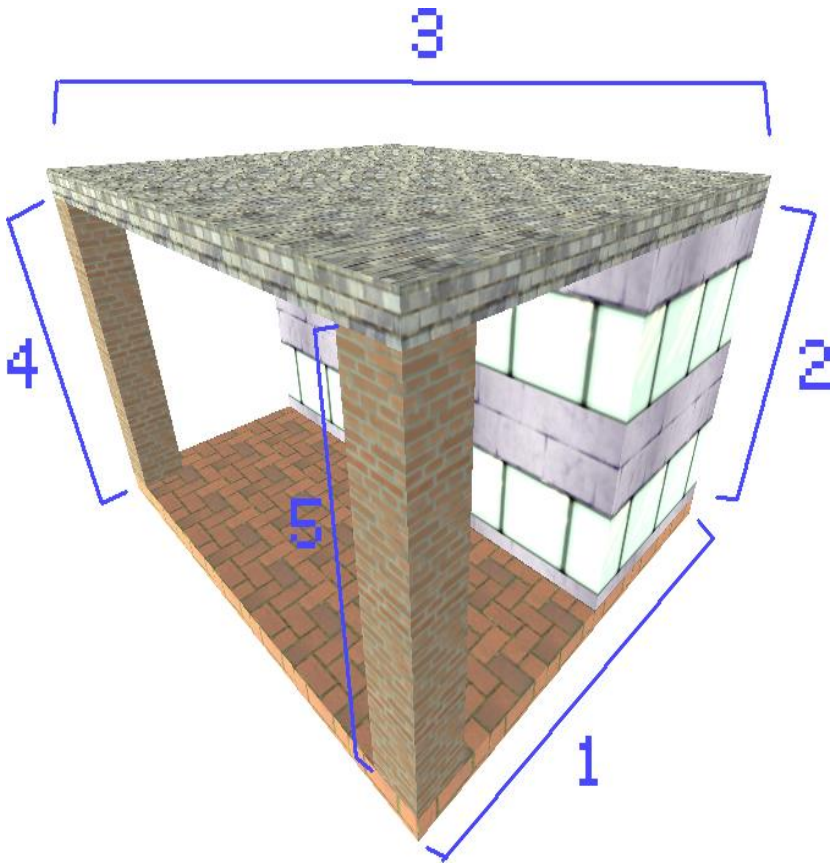
Nota: Se señalan los primitivos que lo conforman así como algunos parámetros

6.5. Sub-generador: cafetería

El sub-generador de cafeterías toma los mismos parámetros que el sub-generador de edificios. Cada cafetería se construye por medio de 5 cuboides (señalados del 1 al 5 en la imagen). Los cuboides 4 y 5 son de tamaño fijo, mientras que las dimensiones horizontales de los demás cuboides van a depender de los parámetros (en particular, de las dimensiones de la base).

No se ofrece la posibilidad de rotar la orientación. En su lugar, el diseño del algoritmo tiene en cuenta la orientación por defecto (apuntando hacia el este) cada vez que se genera una cafetería.

Figura 16. *Apariencia in-engine de una cafetería.*



Nota: Se señalan los primitivos que la conforman.

6.6. Sub-generador: casa

El sub-generador de casas toma los mismos parámetros que el sub-generador de edificios más un parámetro adicional que define la orientación de la fachada como un número entero en el rango [0,3] y cuyo valor se interpreta como una constante simbólica asociada a una dirección cardinal. La forma en como se implementó la rotación para este sub-generador consiste en establecer condiciones que modifican las dimensiones y coordenadas de origen de cada primitivo. El sub-generador se puede separar en 3 partes: el tejado, el bloque central y el área frontal.

El bloque central consiste en un cuboide que representa el cuerpo principal de la casa, cuya textura se escoge aleatoriamente entre una lista de posibles candidatas y su altura se define aleatoriamente. La fórmula para la altura es:

$$h = randfrange(3,4)$$

La función randfrange(a,b) retorna un número real en el rango [a, b] con distribución uniforme.

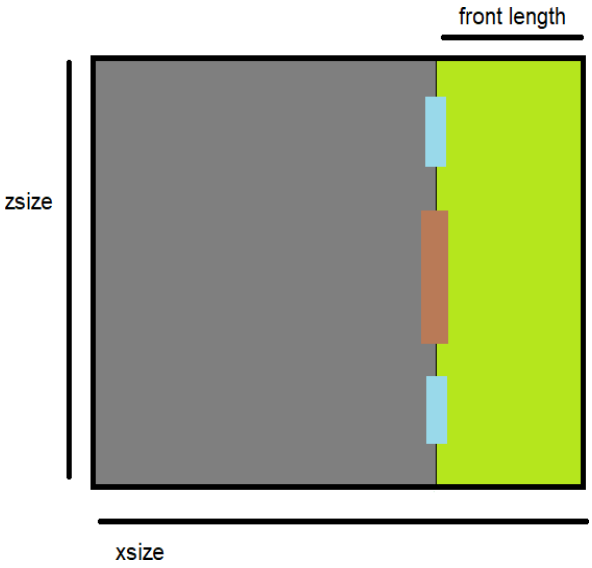
Como adición al bloque central, se genera un cuboide para representar la puerta, entre 1 y 2 cuboides para representar ventanas y un cuboide opcional para representar un banner.

El tejado consiste en seleccionar un número entre 0, 2 y 3 para definir la cantidad de cuboides que lo componen. Posteriormente se generan los cuboides uno encima del otro, disminuyendo las dimensiones de los más elevados pero manteniéndolos concéntricos.

Finalmente, el área frontal consiste en elegir uno de tres estilos. El primer estilo simplemente no genera ninguna geometría. El segundo estilo utiliza un cuboide para representar pasto y otro cuboide para representar un camino de baldosas. El tercer estilo utiliza tres secciones de plano vertical para representar rejas que cubren la fachada.

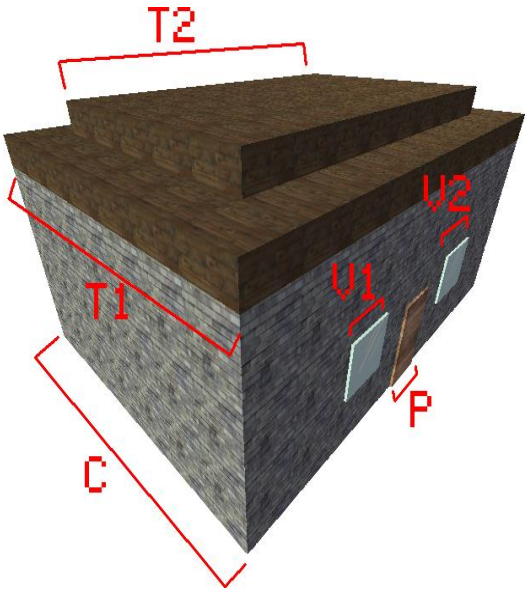
Desde la vista cenital, se define la distribución del espacio, señalando las dimensiones enviadas como parámetro

Figura 17. *Diagrama utilizado en el proceso de diseño del sub-generador de casas.*



Nota: El espacio gris constituye el bloque central. El espacio verde constituye el área frontal. Los rectángulos azules y marrón representan la posición de las ventanas y la puerta respectivamente.

Figura 18. *Apariencia in-engine de una casa.*



Nota: Se señala el bloque central (C), las ventanas (V1, V2), la puerta (P) y el tejado (T1 y T2).

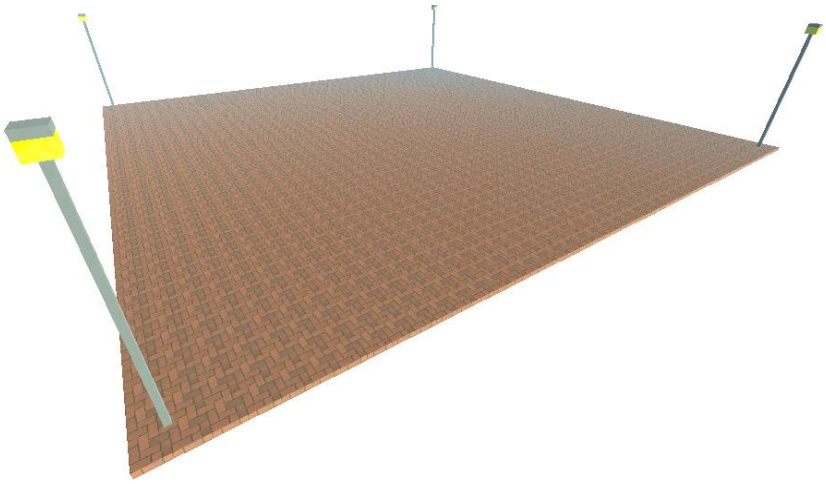
Figura 19. *Apariencia in-engine de tres casas.*

Nota: Se muestran los tres tipos de área frontal y los tipos de tejado posibles. Adicionalmente se muestra la apariencia del banner en caso de que sea generado.

6.7. Sub-generador: andén

El sub-generador de andenes toma los mismos parámetros que el sub-generador de edificios. Cada andén consiste de un cuboide y cuatro postes de luz, uno para cada esquina. La textura para el cuboide se escoge aleatoriamente de una lista de candidatas.

Figura 20. *Apariencia in-engine de un andén.*



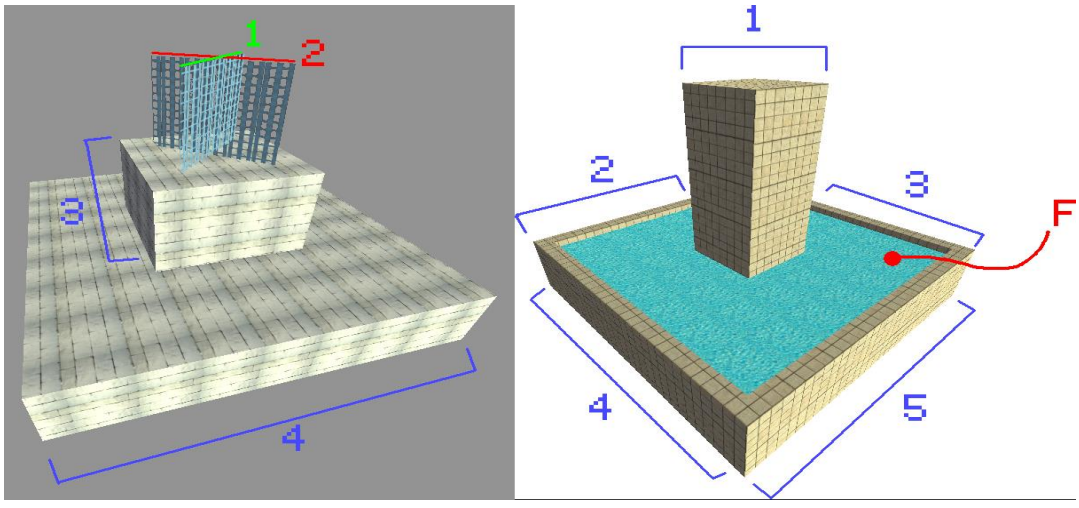
6.8. Sub-generador: estatua

El sub-generador de estatuas toma los mismos parámetros que el sub-generador de edificios. Este sub-generador cuenta con dos diseños posibles, cada uno con un 50% de probabilidad de ser escogido.

El primer diseño consiste de dos cuboides (señalados con 3 y 4) y dos secciones de planos verticales (señaladas con 1 y 2). Las dimensiones horizontales de estos primitivos van a depender de los parámetros enviados, mientras que la altura es constante. La textura utilizada para los cuboides se escoge aleatoriamente de una lista de candidatas, mientras que la textura de los otros primitivos es constante.

El segundo diseño consiste de cinco cuboides (señalados del 1 al 5) y una sección de plano horizontal (señalada con F). De manera similar al primer diseño, las dimensiones horizontales dependen de los parámetros mientras que la altura es constante, y la textura de los cuboides se escoge aleatoriamente mientras que la textura del otro primitivo es constante.

Figura 21. *Apariencia in-engine de ambos diseños del sub-generador de estatuas.*



Nota: Se señalan los primitivos que los conforman (izquierda: primer diseño).

6.9. Contenido universal de los generadores

Todos los generadores comparten un conjunto de parámetros: una coordenada sobre el plano horizontal, dimensiones expresadas como largo y ancho (en algunos casos representadas como un único escalar y en otros como dos escalares) y una semilla para el RNG. Adicionalmente, todos los generadores hacen uso del sub-generador de andenes de manera idéntica: se genera sobre el área especificada por los parámetros antes de realizar cualquier otro procedimiento.

6.10. Generador: manzana normal

Una manzana normal consiste simplemente de casas y edificios residenciales.

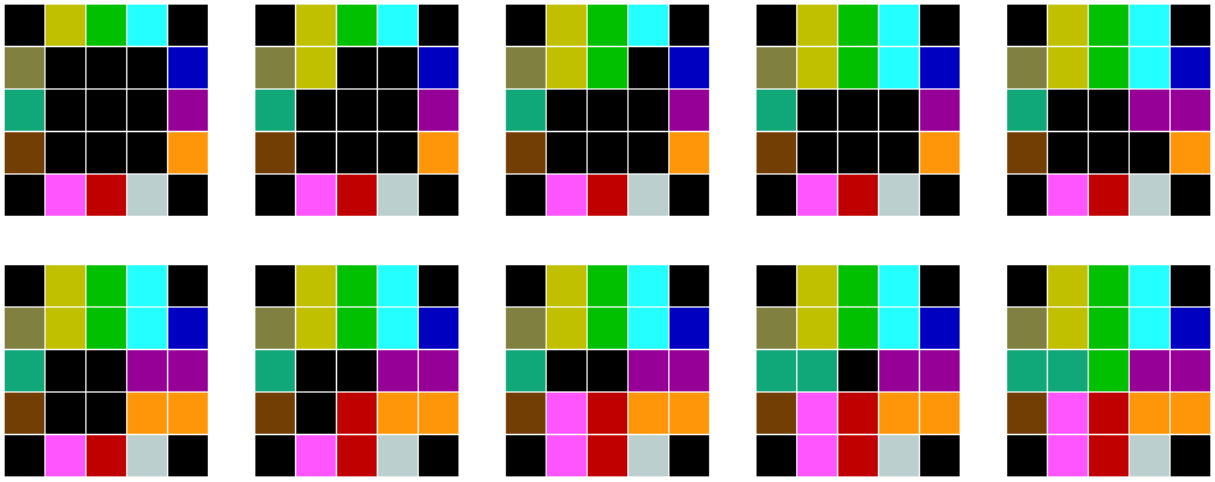
Este generador es el único que cuenta con un parámetro adicional: la densidad de edificios. Esta densidad debe ser un número en el rango [0,1], y representa la probabilidad de reemplazar cada casa por un edificio.

Este generador cuenta con dos etapas. La primera etapa consiste en generar arbustos. Los arbustos simplemente se representan con cuboides con una textura específica, y son generados a lo largo de los cuatro bordes del andén. La segunda etapa consiste en definir las áreas que le corresponde a cada edificación y posteriormente hacer uso de los sub-generadores de casas y edificios. Para definir estas áreas, primero se decide aleatoriamente cuántas fachadas habrá por cuadra (entre 4 y 5). Luego, se utiliza un array 2D donde cada casilla guarda un valor que le corresponde a una edificación específica o un valor nulo que representa que aún no hay ninguna edificación asignada. Inicialmente se definen las casillas de los bordes, asignándoles un valor no nulo y único a cada una, mientras que al resto de casillas se les asigna un valor nulo. Se realiza un proceso iterativo, recorriendo las casillas de los bordes en el sentido de las manecillas de reloj. En este proceso se revisa si es posible propagar el valor de la casilla actual a la casilla aledaña en dirección al centro del array. Si la casilla aledaña tiene

el mismo valor, se repite este proceso manteniendo la misma dirección y saltando a la casilla aledaña. Si la casilla aledaña tiene un valor nulo, se le asigna el mismo valor de la casilla actual y se continúa la iteración. Si la casilla aledaña tiene un valor distinto y no nulo, se continúa con la siguiente iteración. Este proceso continúa hasta que no existan casillas nulas. En la siguiente animación se ilustra el proceso: https://drive.google.com/file/d/1Vr-PjeNLMYMEdGmVJbiqnTfqQrptcd6E/view

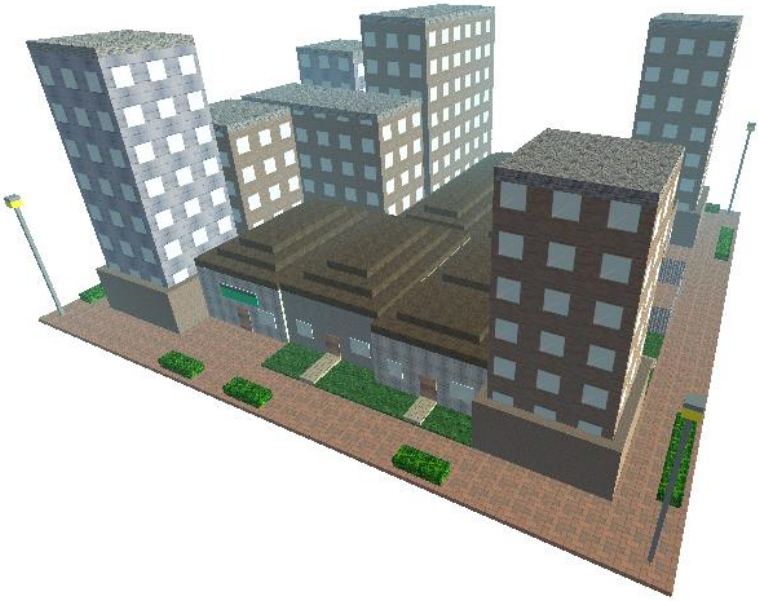
Finalmente, se interpreta este array como la vista cenital de la sección de la manzana que contiene las edificaciones. Para cada valor único en el array, se hace un llamado al RNG en el rango [0,1] y si este es menor a la densidad de edificios se genera un edificio, de lo contrario se genera una casa.

Figura 22. *Ilustración del proceso utilizado para propagar los valores del array en el generador de manzanas normales.*



Nota: Se muestra cómo cambian los valores a lo largo del tiempo (de izquierda a derecha y de arriba a abajo). El color negro representa un valor nulo, el resto de colores representan una referencia a una edificación específica. Las esquinas no se consideran ya que estas nunca pueden propagarse con este modelo.

Figura 23. *Apariencia in-engine de una manzana normal.*



6.11. Generador: centro comercial

Un centro comercial consiste en dos edificaciones conectadas por medio de un puente, una estatua y una pequeña área de pasto con un árbol.

Para definir la ubicación de cada elemento se utiliza un array 2D de 10x10 celdas. El valor de cada celda puede ser nulo, representando que el espacio está disponible, o un número entero que representa una constante simbólica que identifica de manera única a cada elemento.

Inicialmente, se asignan los valores que le corresponden a las edificaciones. Para ello, se definen tres "bloques" (en este contexto, un bloque es una sección rectangular de celdas que comparten el mismo valor). Dos bloques para el primer edificio y un tercer bloque para el segundo edificio. Cada bloque tiene 3 celdas de ancho y altura en función del bloque específico, y se ubica en una posición horizontal específica y una posición vertical aleatoria. Para cada celda de los 3 bloques, se revisa si relativo a las direcciones cardinales cuentan con una celda aledaña cuyo valor sea nulo. En caso de serlo, se cambia el valor de esta celda para reflejar que es parte de la edificación pero también que se requiere generar un pilar en esa posición.

Una vez asignadas las celdas para las edificaciones, se definen las celdas para el puente. Este proceso consiste en encontrar la primera secuencia horizontal de celdas completamente despejada entre el primer y tercer bloque. Una vez encontrada, se marcan todas las celdas con el valor correspondiente al puente.

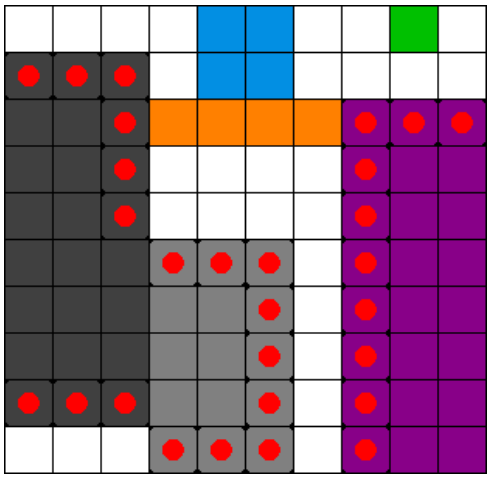
Finalmente, tanto para la estatua como para el árbol se realiza el mismo proceso: se buscan todas las posiciones del array en donde haya espacio disponible para ubicarlas y se escoge una al azar.

Tras completar la configuración del array, se inicia el proceso de generar geometría. Para las edificaciones, se define una parte baja y alta. La parte baja simplemente consiste de un cuboides que cubren cada celda. En particular, para aquellas celdas marcadas como pilares, se crea un cuboide concéntrico a la celda de poca longitud. Para el resto de celdas el cuboide cubre toda el área. La parte alta consiste de tres cuboides de dimensiones horizontales idénticas apilados verticalmente, donde el cuboide en el medio es considerablemente más alto y cuenta con una textura distinta. Adicionalmente, se crea un cuboide con textura de banner en la pared oriente de la segunda edificación.

Para el puente se utilizan tres cuboides y dos secciones de plano vertical.

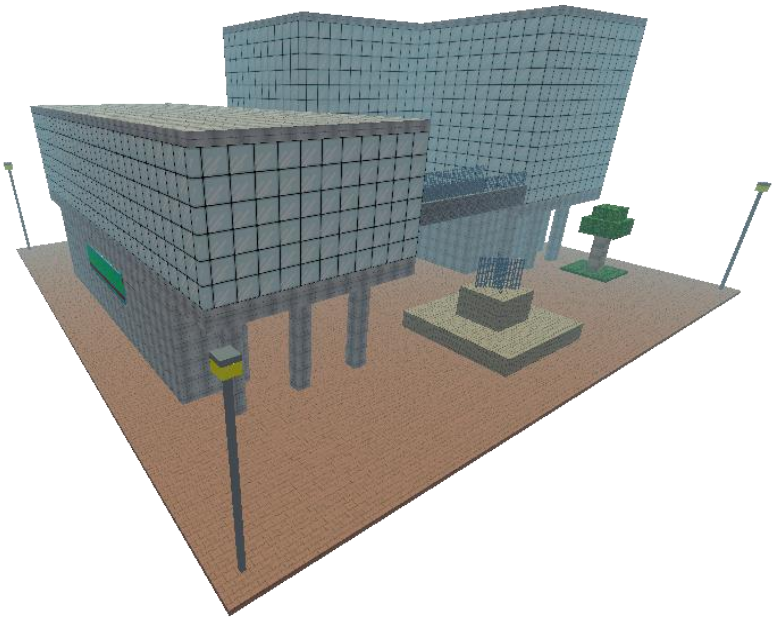
Para el pasto se utiliza un cuboide de poca altura. El resto de la geometría simplemente hace uso de los sub-generadores.

Figura 24. *Diagrama demostrando una de las posibles configuraciones para el estado final del array del generador de centros comerciales.*



Nota: Las celdas blancas son nulas. Las celdas grises y moradas representan las edificaciones. Los puntos rojos representan los pilares. Las celdas naranjas representan el puente. Las celdas azules representan la estatua y la celda verde representa el árbol.

Figura 25. *Apariencia in-engine de un centro comercial.*



6.12. Generador: parque

Un parque consiste en una base de pasto, un sendero, una estatua, bancas y árboles.

Para definir la ubicación de cada elemento se utiliza un array 2D de 7x7 celdas. Inicialmente, se establece el valor de cada celda como nulo.

Las primeras celdas que se marcan son aquellas que identifican al sendero. Para este fin, primero se escogen dos celdas en lados opuestos del array, garantizando que no compartan ninguna coordenada. Estas celdas se consideran los extremos del sendero. Adicionalmente se define N, un número entero aleatorio en el rango [2,6], el cual se interpreta como número de celdas antes de desviar el sendero. Partiendo desde uno de los extremos y moviéndose en dirección al otro extremo, se van marcando las celdas con el valor correspondiente al sendero. Una vez se marcan N celdas, se realiza un proceso similar a lo largo del otro eje, con el objetivo de alinear el sendero con una coordenada. Tras alinear el sendero se cambia nuevamente el eje de movimiento y se marcan las celdas hasta llegar al otro extremo.

La segunda etapa es la que se encarga de definir la ubicación de la estatua. Sin tener en cuenta las celdas en los bordes, se revisan todas las posiciones de 2x2 celdas donde hay espacio libre. Se escoge una de estas posiciones al azar y se marcan las 4 celdas como pertenecientes a la estatua.

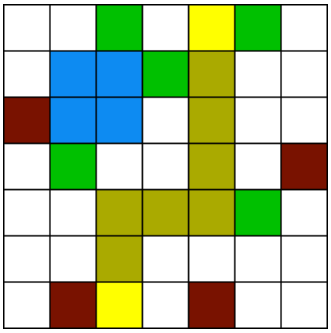
La tercera etapa consiste en definir qué celdas contienen bancas. Para ello simplemente se hacen 6 intentos. En cada intento se escoge aleatoriamente una celda sobre el borde de array y si está disponible, se marca la celda.

Finalmente, para definir las posiciones de los árboles simplemente se recorre secuencialmente el array y para cada celda disponible hay una probabilidad de marcarla.

Al terminar la configuración del array, se procede con la generación de geometría. Primero, para representar el pasto se utiliza un cuboide concéntrico al andén que cubre una porción grande de la manzana. Para el sendero se utiliza un algoritmo que identifica celdas aledañas que se puedan agrupar

en un rectángulo donde todas comparten el mismo valor. Para cada rectángulo identificado, se genera un cuboide. El resto de la geometría hace uso de props y sub-generadores.

Figura 26. *Diagrama mostrando una de las posibles configuraciones para el estado final del array del generador de parques.*



Nota: Las celdas blancas son nulas. Las celdas amarillas representan el sendero y los extremos se resaltan con más brillo. Las celdas azules representan la estatua. Las celdas marrones representan las bancas. Las celdas verdes representan los árboles.

Figura 27. *Apariencia in-engine de un parque.*

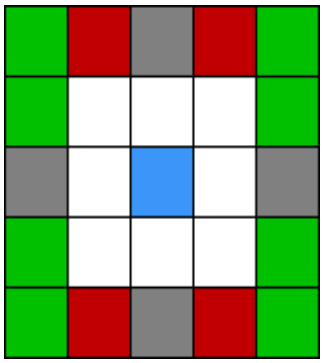
6.13. Generador: conjunto residencial

Este generador cuenta con dos diseños posibles, de los cuales se elige uno al azar. En general, un conjunto residencial consiste en una estatua, varias edificaciones y algunas rejas. Sin embargo los diseños cuentan con algunas diferencias en sus elementos geométricos.

El primer diseño consiste en cuatro edificios, cuatro rejas y una estatua. La ubicación y dimensiones de cada elemento respecto al plano horizontal son constantes para este diseño, así que si bien se utilizó un array 2D para definir conceptualmente los planos del lugar, no es necesario instanciar un array ni calcular valores para celdas. La aleatoriedad para la geometría de este diseño viene entonces de la variación natural de los sub-generadores, la textura utilizada para las edificaciones y la altura de estas.

Para la generación de geometría, primero se utilizan dos ciclos for cuyo comportamiento es similar: generar cuboides apilados verticalmente con las mismas dimensiones pero distintas coordenadas. El cuboide generado en el centro de la pila utiliza una textura diferente a los demás. Luego, por medio de secciones de plano vertical, se generan las rejas que conectan la base de las edificaciones. Finalmente se utiliza el sub-generador de estatuas en el centro de la manzana. En la figura 19 se muestra un plano simplificado de la vista cenital del primer diseño.

Figura 28. *Diagrama utilizado en el proceso de diseño del generador de conjuntos residenciales.*



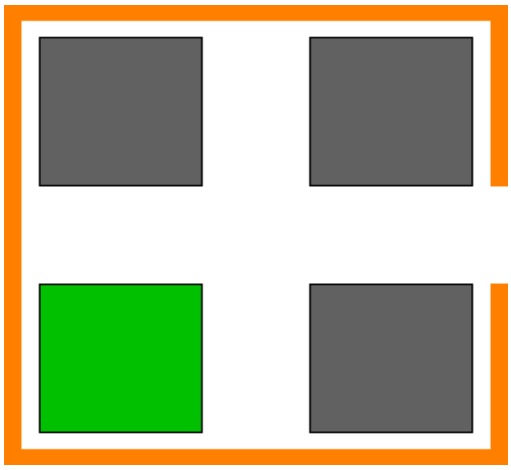
Nota: Las celdas verdes y rojas le corresponden a los cuboides de las edificaciones, siendo cada color el que identifica qué ciclo for se encarga de generarlo. La celda azul representa la estatua. Las celdas grises representan los lugares donde se generan las rejas.

El segundo diseño consiste en una sección con pasto y una estatua, tres edificios y un borde de ladrillos y rejas. La sección de pasto tiene las mismas dimensiones horizontales que la base de cada

edificio. Se definen cuatro áreas espaciadas de manera uniforme, donde una de ellas aleatoriamente va a contener la sección de pasto. Las tres áreas restantes van a contener los edificios. El borde de ladrillos y rejas envuelve el lugar, dejando una entrada por el lado oriente de la manzana.

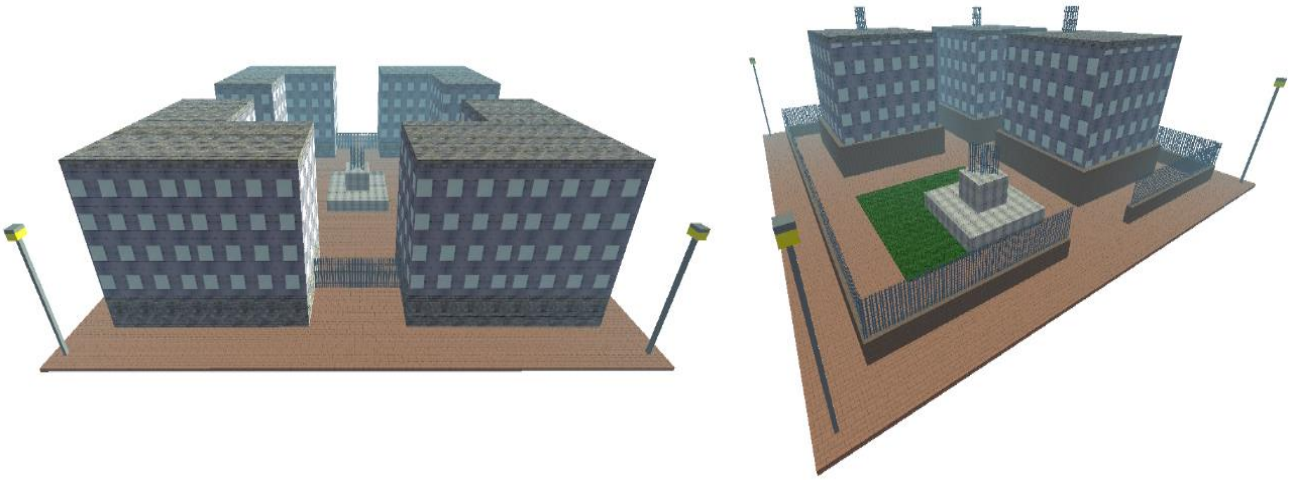
Respecto a la generación de geometría, primero se generan los bordes. La parte de ladrillo utiliza un cuboide, mientras que las rejas utilizan secciones de plano vertical. Posteriormente, se recorre cada una de las cuatro áreas. En el caso donde se deba generar la sección de pasto, se utiliza un cuboide de poca altura para representar el pasto y se invoca el sub-generador de estatuas. En el caso contrario, simplemente se utiliza el sub-generador de edificios, asegurándose de que todos los edificios se generen utilizando la misma semilla de manera que tengan una apariencia uniforme.

Figura 29. *Diagrama utilizado en el proceso de diseño del generador de conjuntos residenciales.*



Nota: Se muestra un plano simplificado de la vista cenital del segundo diseño. El polígono naranja representa el borde. Los cuadrados representan las cuatro áreas, una de las cuales se colorea distinto para representar la sección de pasto.

Figura 30. *Apariencia in-engine del primer (izquierda) y segundo (derecha) diseño del generador de conjuntos residenciales.*



6.14. Generador: hospital

Un hospital consiste en una gran edificación y una zona verde rodeada por un borde de ladrillos y rejas que contiene pasto, árboles y una estatua.

La posición y dimensiones horizontales de la base de la edificación y la zona verde son constantes. Sin embargo, la posición de los contenidos de la zona verde es aleatoria, así como la elevación y las dimensiones de los pisos superiores de la edificación.

Para definir la ubicación de los elementos de la zona verde, se utiliza un array 2D de 3x3 celdas que representa una versión simplificada de la vista cenital. Inicialmente se asigna un valor nulo a todas las celdas. Posteriormente, se elige una sección de 2x2 celdas en una posición aleatoria y se marcan estas celdas como parte de la estatua. Finalmente, se recorre el array y para cada celda nula se establece una probabilidad de 0.3 para definir la celda como contenedora de un árbol.

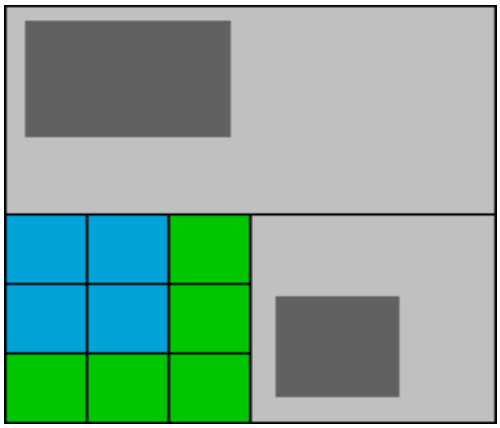
El proceso de modelado inicia con la edificación. Se define una función que genera cuatro cuboides apilados. Las dimensiones horizontales de los dos primeros cuboides son idénticas, mientras que las dimensiones horizontales de los últimos son menores e iguales entre sí y su área horizontal está contenida dentro del área horizontal de los primeros cuboides. La altura del primer cuboide se define como:

$$h = 3 * randirange(4,5)$$

La altura del segundo y cuarto cuboide son constantes, mientras que la altura del tercer cuboide es $h/4$. Las texturas son deterministas y se utilizan dos, siendo intercaladas entre cada cuboide. Esta función es llamada dos veces, en la primera se cubre la mitad superior de la manzana, mientras que en la segunda se cubre la esquina inferior derecha.

En el área restante de la manzana se crea el jardín. Para el pasto y borde de ladrillos se utilizan cuboides de dimensiones, posición y textura determinista. Para las rejas metálicas se utilizan secciones de plano vertical, nuevamente con posición y dimensiones deterministas. El resto de la geometría es creada por medio de props y sub-generadores.

Figura 31. *Diagrama utilizado en el proceso de diseño del generador de hospitales.*



Nota: Se muestra un plano simplificado de la vista cenital del hospital. Las áreas grises representan la edificación, siendo las de color oscuro los cuboides de mayor elevación. Las celdas azules representan la estatua, mientras que las celdas verdes representan el resto de la zona verde.

Figura 32. *Apariencia in-engine de un hospital.*

6.15. Generador: instituto

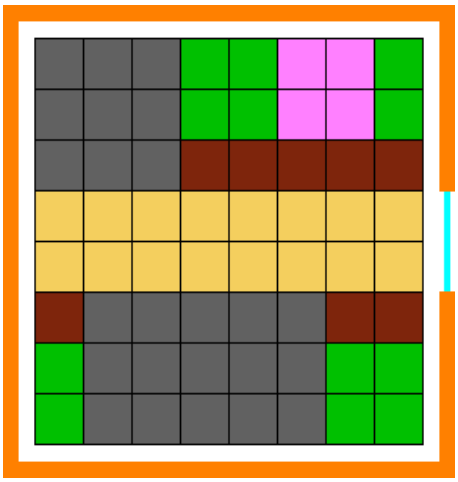
Un instituto consiste en un borde de ladrillos, una reja, dos edificios, una cafetería, un camino central, bancas, zonas de pasto y árboles.

La reja y el borde de ladrillos envuelven al resto de la geometría, ambos con dimensiones y posiciones deterministas. El resto de los elementos dependen de un array 2D de 8x8 celdas para definir su ubicación.

Inicialmente, se marcan todas las celdas del array como nulas excepto las dos filas centrales, las cuales son marcadas como parte del camino central. Posteriormente, se define una sección rectangular de celdas para cada edificio. Las dimensiones y posición vertical en la cuadrícula son constantes, mientras que la posición horizontal se elige aleatoriamente. Luego, se calculan todos los espacios de 2x2 celdas nulas y se escoge uno aleatoriamente, cuyas celdas le corresponderán a la cafetería. Después se les asignan a todas las celdas nulas aledañas al camino central el valor correspondiente a una banca. Finalmente, se recorre el array y para cada celda nula se establece una probabilidad de 0.4 de generar un árbol.

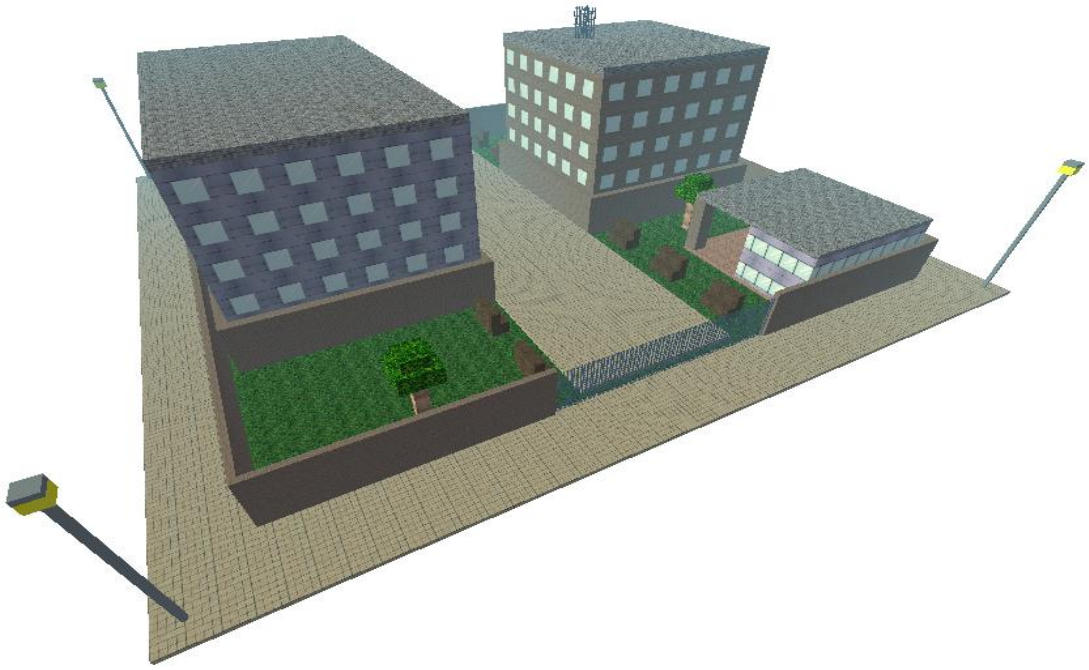
Para la generación de geometría se inicia con los bordes. Se utilizan cuboides para los ladrillos y secciones de plano vertical para las rejas. Al interior se utiliza un cuboide de poca altura con textura de pasto que cubre toda el área. Los edificios y la cafetería hacen uso de sus respectivos sub-generadores, mientras que las bancas y árboles simplemente se generan por medio de sus respectivos props.

Figura 33. *Diagrama utilizado en el proceso de diseño del generador de institutos.*



Nota: Se muestra un plano simplificado de la vista cenital del instituto. El área naranja y azul le corresponde al borde y la reja respectivamente. Las celdas verdes son nulas. Las celdas grises les corresponden a los edificios. Las celdas amarillas representan el camino central. Las celdas marrones representan las bancas. Las celdas rosadas representan la cafetería.

Figura 34. *Apariencia in-engine de un instituto*



7. Resultados

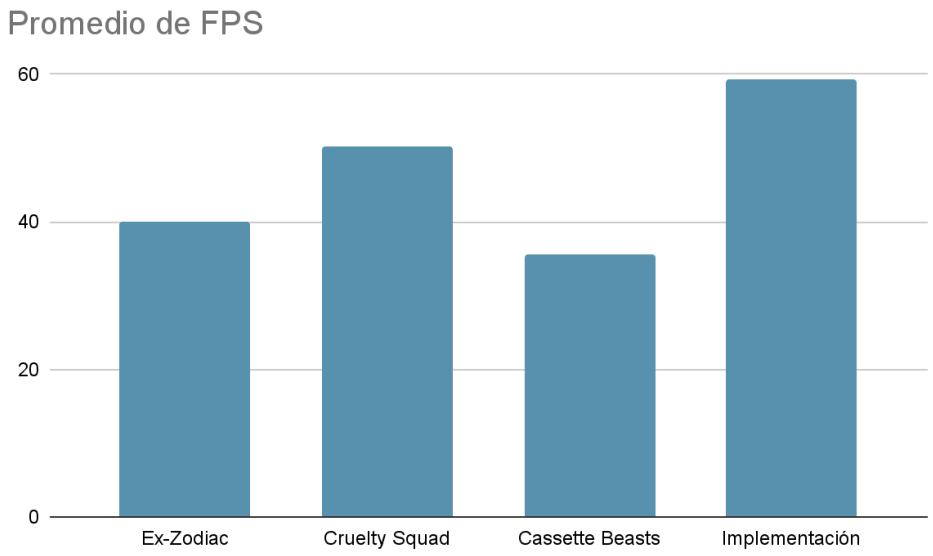
7.1. Validación del rendimiento

Son de interés dos métricas: fotogramas por segundo (FPS) y tiempo de generación. Los FPS van a depender de la eficiencia de la visualización dinámica en tiempo real, mientras que el tiempo de generación va a depender de la eficiencia de la implementación del algoritmo de generación procedural. Para los FPS un valor alto es mejor. Para el tiempo de generación un valor bajo es mejor.

Ya que el interés del proyecto tiene que ver con desarrollar habilidades prácticas para la industria de los videojuegos, se decidió que la validación de las métricas debe definirse por medio de una comparación entre la implementación del algoritmo y productos comerciales de la industria. Para este fin, se diseñaron dos pruebas con un formato similar:

Prueba de FPS. Se comparan los FPS de la implementación y tres videojuegos comerciales en 3D desarrollados en Godot, cuya fidelidad de modelado es similar a la de este proyecto. Cada uno se ejecuta sin ningún otro programa simultáneamente, con las opciones gráficas en una configuración baja y con la visualización de FPS activada. Durante 20 segundos se anota la secuencia de valores de FPS en la primera escena 3D. Posteriormente se calcula el promedio de esta secuencia. Los resultados se presentan a continuación:

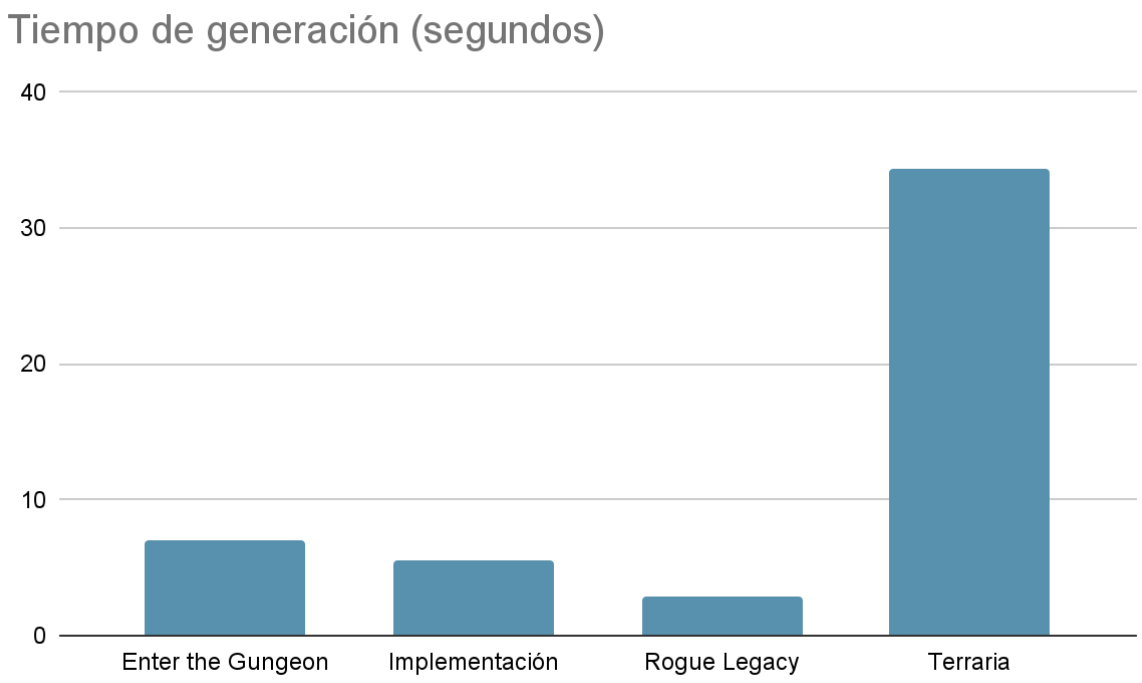
Figura 35. *Diagrama mostrando los resultados del promedio de FPS.*



Prueba de tiempo de generación. Se compara el tiempo de generación de la implementación y tres videojuegos comerciales con generación procedural de escenarios. Cada uno se ejecuta sin

ningún otro programa simultáneamente. Para la implementación se escogieron los parámetros más costosos, mientras que para los videojuegos se eligieron los parámetros menos costosos en los casos donde se permite la parametrización. Se graba un video del proceso de generación. Con el fin de no alterar el rendimiento de cada programa, en lugar de grabar la pantalla por medio de software se utilizó una cámara externa. Posteriormente, la grabación se analiza en un editor de video, recortando los fotogramas exactos en donde inicia y finaliza el proceso de generación. Con estos recortes y el editor de video es posible ver el tiempo exacto que tarda la generación en la grabación. A continuación se presentan los resultados:

Figura 36. *Diagrama mostrando los resultados del tiempo de generación.*



Una aclaración importante es que muy difícilmente se pueden encontrar dos videojuegos con generación procedural de escenarios cuyo algoritmo sea similar en términos de geometría, complejidad y técnicas utilizadas. Por tanto, estas comparaciones no pueden establecer la calidad de la implementación para el tipo específico de escenarios deseados. En su lugar, desde el punto de vista de las expectativas del mercado, estas pruebas ayudan a establecer si existe algún defecto importante con la implementación.

8. Conclusiones

Teniendo en cuenta la cantidad de manzanas que pueden ser generadas, la cantidad de primitivos por manzana y la necesidad de renderizar en tiempo real, se hace prioritario establecer subsistemas que optimicen el rendimiento de la implementación. Este hecho fue algo que se hizo evidente en las primeras etapas de proyecto, donde previo a la implementación de la cuadrícula de chunks se podía notar una caída de rendimiento en función del tamaño de la ciudad.

Las funciones generadoras de primitivos son los cimientos del algoritmo. Sus limitaciones son aquello que define qué tipo de escenarios es posible generar. Adicionalmente, desde un punto de vista de legibilidad, eficiencia y mantenibilidad de software, es importante encontrar un balance entre la cantidad de parámetros que requiere cada primitivo y su flexibilidad para representar distintas formas.

Las distintas técnicas estudiadas para la generación procedural de contenido pueden utilizarse simultáneamente y sinergizar entre sí. Por ejemplo, en este proyecto se puede apreciar como los generadores utilizan tanto el método paramétrico como la representación intermedia de un escenario en 3D por medio de un plano de la vista cenital.

Al analizar tanto el generador procedural presentado en este proyecto como otros existentes en la literatura, es posible notar que una limitación ubicua de la generación procedural es la dificultad para diseñar un algoritmo que a partir de distintas semillas sea capaz de generar geometrías con disparidad en sus características generales. En la opinión del autor, el núcleo del problema consiste en que diseñar un sistema con múltiples propiedades emergentes no es trivial.

9. Trabajo futuro

Es posible generar mayor variedad de contenido simplemente con incluir nuevos generadores, sub-generadores y props.

Algunos de los sub-generadores y generadores cuentan con múltiples diseños. Es posible expandir este concepto para cualquier procedimiento que genere geometría.

Parte de la geometría generada está ubicada de tal manera que desde ningún ángulo de cámara será visible (por ejemplo, la mayoría del cuboide principal de los andenes se ve cubierta por otros objetos). Si bien esto se hizo por simplicidad, es posible omitir esta geometría para incrementar el rendimiento.

Las ciudades generadas carecen completamente de lugares subterráneos. Incluir tal geometría sería un desafío interesante, ya que el modelo de manzanas y generadores no sería suficiente para satisfacer este requerimiento.

Gran parte de la geometría generada carece de la capacidad de rotarse. En general, la rotación solo se presenta como opción en los casos donde es absolutamente necesario. Para implementar rotaciones generales, la solución ideal requiere refactorizar la funcionalidad que permite generar primitivos.

El algoritmo hace uso extensivo de arrays 2D como herramientas de representación intermedia. Sin embargo, se podrían explorar nuevos generadores que utilicen otras estructuras de datos, como grafos o arrays 3D.

Se podrían ignorar algunas de las limitaciones establecidas como compromiso en la especificación inicial de geometría, tales como modelado al interior de las edificaciones, variación en el nivel del suelo, vías de mayor complejidad y modelado de mayor fidelidad.

1. Referencias

Acuerdo 4 de 2007 [Consejo Superior de la Universidad Industrial de Santander]. Por el cual se modifica el Reglamento Académico Estudiantil de Pregrado, en su Título V, Capítulo IX "Del Trabajo de Grado". 12 de febrero de 2007.

Adaptado de "Usando Tilemaps". Godot Docs, (s.f.). (https://docs.godotengine.org/en/stable/tutorials/3d/introduction_to_3d.html) CC-BY 3.0 [2014] por Juan Linietsky, Ariel Manzur y la comunidad de Godot.

Andrade, A. (2015). Game engines: a survey. *EAI Endorsed Transactions on Serious Games.* DOI: 10.4108/eai.5-11-2015.150615

Barriga, Nicolas A. (2018). A Short Introduction to Procedural Content Generation Algorithms for Videogames. DOI:10.1142/S0218213019300011

Chen, J. X. (2003). *Guide to Graphics Software Tools*. Springer New York, *NY*. DOI: 10.1007/b97643

Compton, K., & Mateas, M. (2021). Procedural Level Design for Platform Games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. DOI: 10.1609/aiide.v2i1.18755

Dong, J. Liu, J. Yao, K. Chantler, M. Qi, L. Yu, H. Jian, M. (2020) Survey of Procedural Methods for Two-Dimensional Texture Generation. *Sensors*. DOI: 10.3390/s20041135

FileFormat.Info (s.f).Wavefront OBJ File Format Summary. Recuperado de https://www.fileformat.info/format/wavefrontobj/egff.htm

Freiknecht, J. Effelsberg, W. (2017). A Survey on the Procedural Generation of Virtual Worlds. *Multimodal Technologies and Interaction*. 2017. DOI: 10.3390/mti1040027

Godot (s.f) *License*. https://godotengine.org/license/

Johnston, D. (2018). *Random Number Generators--Principles and Practices: A Guide for Engineers and Programmers*. Boston: DEG Press, 2018. Web. DOI: 10.1515/9781501506062

Kelly, G. McCabe, H. (2006) A Survey of Procedural Techniques for City Generation, *The ITB Journal*: Vol. 7: Iss. 2, Article 5. DOI:10.21427/D76M9P

Linietsky, J. (26 de febrero de 2021) Why isn't Godot an ECS-based game engine? GODOT. (https://godotengine.org/article/why-isnt-godot-ecs-based-game-engine/)

Mark, H. Meijer, S. Van Der Velden, J. Iosup, A. (2013) Procedural Content Generation for Games: A Survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) DOI: 10.1145/2422956.2422957

Marschner, S. & Shirley, P. (2016). Fundamentals of Computer Graphics (4th ed.). A K Peters/CRC Press. DOI: 10.1201/9781315372198

Short, T., & Adams, T. (2017). Procedural Generation in Game Design (1st ed.). A K Peters/CRC Press. DOI: 10.1201/9781315156378

Togelius, J. Whitehead, J. Bidarra, Rl. (2011). Procedural Content Generation in Games (Guest Editorial). Computational Intelligence and AI in Games. IEEE Transactions on Computational Intelligence and AI in Games. DOI: 10.1109/TCIAIG.2011.2166554

Yu, D. (2016). *Spelunky.* Gabe Durham. Boss Fight Books, Los Angeles, CA.

Adaptado de "Introduction to 3D". Godot Docs, 2014 (https://docs.godotengine.org/en/stable/tutorials/3d/introduction_to_3d.html) CC-BY 3.0 [2014] por Juan Linietsky, Ariel Manzur y la comunidad de Godot.