UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

# Creación de una extensión de Unity 3D para la generación procedural de terreno

Trabajo Fin de Grado

Grado en Ingeniería Informática


AUTOR/A: Turró Alcalá, Carlos Javier

Tutor/a: Lluch Crespo, Javier



CURSO ACADÉMICO: 2023/2024

# Resumen

El objetivo del siguiente trabajo de fin de grado es investigar y entender las diferentes herramientas y metodologías disponibles para la generación procedural de contenido, basada en la idea de números pseudoaleatorios. En el paradigma de la generación procedural actualmente no existe una única herramienta para controlar la misma en todos sus aspectos, así que será necesario la creación de una que se adecúe a las necesidades del proyecto. Si bien no existe dicha herramienta, si son bien conocidas diversas técnicas como el uso de Ruido Perlin, de gramáticas o de sistemas paramétricos. Para lograr el objetivo se planteará el desarrollo de dos de estas técnicas para la generación de un terreno decorado con estructuras, ordenado de la siguiente manera:

- En primera instancia, se realizará el diseño y desarrollo de un generador de terreno basado en Ruido Perlin con diferentes tipos de terrenos en función del valor del propio ruido.
- En segundo lugar, también se diseñará y desarrollará un generador de estructuras fundamentado en una gramática predefinida y estas estructuras serán distribuidas sobre el terreno.
- Finalmente, con ambos generadores se hará una demo en la cual se podrán cambiar diferentes parámetros para comprobar diversas configuraciones.

Para la realización de todo el proyecto se usará la plataforma Unity, dadas las facilidades que ofrece a la hora de crear y manejar objetos y de visualizar los mismos en un entorno 3D. Como ya se comentó anteriormente, en Unity no existe ninguna herramienta específica o genérica oficial para la creación procedural de contenido, sino que son los miembros de la comunidad quienes se crean sus propias herramientas. Cualquier referencia a estas herramientas será comentada. Dado que se usará Unity como plataforma, el lenguaje sobre el que se realizará el proyecto será C#, con las debidas órdenes y librerías específicas de Unity.
**Palabras clave:** Procedural, Unity, C#, Gráficos, Terreno

# Abstract

The objective regarding the following degree thesis is to investigate and understand the different methods and tools available for procedural content generation, based on pseudo-random numbers. In procedural generation paradigm, there isn't a tool which controls all of it's possibilities, therefore a new tool which fulfills the project's necessities will need to be developed. For such task, techniques such as Perlin noise, use of grammars or parametric approaches could be used. For the objective of using two of these techniques for the generation of a terrain with structures, the development will be organized as such:

- Firstly, the design and development of a terrain generator based on Perlin noise with different types of terrain based on the noise's values.

- Secondly, the design and development of a building generator based on a predefined grammar, and the distribution of these structures on the terrain.
- Finally, with both generators a demo will be developed where the parameters may be changed to try different configurations.

For the development of the development of the project the Unity engine will be used due to the many tools it offers for creating, modifying and visualize 3D objects. As it was commented before, there is no generic procedural content generation tool in Unity, it's the community members which develop the different libraries. Any reference to any of these libraries will be commented. Since the Unity engine will be used as the chosen platform, any code in the project will be developed in C#, using it's own libraries and methods.

**Key words:** Procedural, Unity, C#, Graphics, Terrain

# Índice general

**Índice general** v

**1 Introducción** **1**
1.1 Motivación . . . 1
1.2 Objetivos . . . 2
1.3 Estructura de la memoria . . . 2

**2 Estado del arte** **5**
2.1 Técnicas . . . 5
2.1.1 Números pseudo-aleatorios . . . 5
2.1.2 Ruido perlin . . . 6
2.1.3 Gramáticas BNF . . . 6
2.2 Terreno . . . 8
2.3 Laberintos y mapas . . . 9
2.4 Producciones gramaticales . . . 10
2.5 Otras producciones . . . 12
2.6 Crítica al estado del arte . . . 12
2.7 Propuesta . . . 13

**3 Decisión de una solución** **15**
3.1 Identificación y análisis de soluciones posibles . . . 15
3.2 Solución propuesta . . . 16
3.3 Plan de trabajo . . . 16

**4 Diseño de la solución** **19**
4.1 Arquitectura del sistema . . . 19
4.2 Diseño Detallado . . . 21
4.2.1 Terreno . . . 21
4.2.2 Construcciones . . . 22
4.2.3 Generador combinado . . . 24
4.2.4 Demo . . . 25
4.3 Tecnología utilizada . . . 25
4.3.1 Requisitos . . . 25
4.3.2 Elecciones . . . 26

**5 Desarrollo de la solución propuesta** **29**
5.1 Preparación . . . 29
5.2 Generador de terreno . . . 30
5.3 Generador de construcciones . . . 36
5.4 Generador combinado . . . 43
5.5 Demo completa . . . 48

**6 Pruebas** **51**
6.1 Corrección . . . 51

6.2 Calidad . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
**7 Conclusiones** **57**
7.1 Relación del trabajo con estudios cursados . . . . . . . . . . . . . . . 58
**8 Trabajo futuro** **59**
**Bibliografía** **61**

---

Apéndice

**A Objetivos de desarrollo sostenible** **63**

# CAPÍTULO 1
# Introducción

En el mundo en el que vivimos nada es infinito. No poseemos de todo el tiempo ni tenemos recursos ilimitados y esto es una verdad ineludible. En el mundo de la informática esta cuestión también es una realidad. Muchas veces se pretende crear entornos o herramientas, pero estas llevan mucho trabajo y tiempo para hacerlas, trabajo y tiempo que no se tienen. Es por ello que muchas veces todas estas ideas recurren a alternativas para cumplir su objetivo. Una de esas alternativas es la generación procedural. Imagine que necesita de dibujar un árbol, es una tarea sencilla y fácilmente conseguible. Ahora imagine que necesita dibujar 100 árboles. La tarea sencilla se vuelve en una larga y repetitiva, sin embargo, uno puede detectar una serie de reglas o procedimientos que se toman en la misma. Ahí es donde entra la generación procedural. En lugar de dibujar los 100 árboles a mano, se podría crear un generador procedural de árboles, a los que, enseñándole la estructura de un árbol genérico, podrían crear estos 100 árboles o incluso mil. Este es el verdadero potencial de la generación procedural, ahorrar tiempo y recursos en tareas que, necesariamente, son repetitivas y toman de muchos recursos humanos.

## 1.1 Motivación

En cuanto a la motivación de la cual surge este trabajo, por un lado yo vi una oportunidad en el desarrollo del trabajo de fin de grado para poder investigar en temas de interés personal como es la generación procedural. Llevado por un interés propio por el desarrollo de videojuegos y una larga trayectoria utilizando el motor de Unity, planteé la posibilidad de realizar el trabajo relacionándolo con la materia de gráficos estudiada. Por otro lado, es clara la necesidad en la comunidad para el desarrollo de herramientas como la planteada, dada la carencia de las mismas. En este contexto, la creación de una nueva facilidad para toda clase de desarrolladores sería tanto positiva como aceptada por dichos usuarios.

## 1.2 Objetivos

El presente trabajo tiene como objetivo principal el desarrollo de una serie de herramientas en forma de extensión de Unity, centradas en la generación procedural de diferentes elementos. El desarrollo de estas herramientas debe permitir un posterior uso de las mismas de manera fácil y eficaz, pudiendo ser utilizadas con mínimo conocimiento de las librerías. A su vez, se pretende recopilar varias técnicas diferentes de generación procedural y combinarlas para demostrar las capacidades de las mismas en una demo interactiva. Para concretar serían:

- Investigar sobre diferentes técnicas de generación procedural y seleccionar varias de ellas.
- Desarrollar para cada una de esas técnicas un generador procedural.
- Combinar todas las técnicas en un único generador.
- Desarrollar una aplicación demo sobre la cual poder visualizar y probar los resultados.

## 1.3 Estructura de la memoria

A continuación, se mostrarán y explicarán todas las cuestiones surgidas de tratar cumplir los objetivos recientemente definidos.

Primeramente se explicará el estado del arte de la generación procedural en todas sus formas y se explicará criticando este mismo estado del arte el porqué de la existencia de este trabajo. En dicho apartado del estado del arte se mostrarán una serie de técnicas utilizadas comúnmente en la generación procedural, así como algunos ejemplos ya existentes. Una vez definida la necesidad del mismo se presentará una propuesta para solucionar esta necesidad.

Antes de poder comenzar a desarrollar la misma, deberá ser necesario analizar detenidamente el problema, las oportunidades posibles y de todas ellas seleccionar una. Finalmente, antes de comenzar el desarrollo será interesante un apartado organizativo en la forma de un plan de trabajo para certificar que se desarrolla acorde al tiempo establecido.

Una vez propuesta la solución, se presentará el diseño y arquitectura de la misma, de manera más detallada que en los puntos anteriores. Este diseño pretende explicar todo lo necesario para no dejar espacio a la duda sobre el sistema que se va a desarrollar. Así mismo, también se comentará sobre todas las tecnologías y librerias utilizadas en el proceso. Dado que la mayor parte de ellas serán externas al desarrollo del TFG, se referenciarán y explicará claramente qué parte de cada una se utilizará en el desarrollo.

Tras ello, una larga sección explicará todo el proceso de desarrollo y todas las decisiones tomadas en el mismo. Dadas las diferentes partes del trabajo que se propone, esta sección indagará profundamente en el proceso detrás de cada una de ellas, mostrando los problemas aparecidos y las decisiones tomadas en el proceso.

Después, se cubrirá en otra sección la fase de pruebas sobre la extensión finalizada y la demo desarrollada. Estas pruebas consistirán por una parte en el correcto funcionamiento de todo el sistema, mientras que en otra se analizará el resultado generado de manera más cualitativa y subjetiva.

Para finalizar, en las conclusiones se hará una recopilación de las decisiones tomadas y las conclusiones a las que se ha llegado, como de los problemas surgidos, los objetivos alcanzados y todo aquello que se ha aprendido en el proceso de desarrollo.

Adicionalmente, se realizará una sección de trabajo futuro, analizando toda clase de oportunidades no llevadas a cabo en el desarrollo de este TFG. Entre ellas se incluirán tanto ideas no realizadas por falta de medios, como ideas surgidas muy tarde en el desarrollo como para ser implementadas y aquellas no realizadas por falta de tiempo.

Cabe destacar que todos los apuntes al código desarrollado en el trabajo serán escritos en inglés en las figuras, mientras que serán explicados en español en la memoria.

# CAPÍTULO 2
# Estado del arte

La generación procedural de contenido es un campo con muchas posibilidades para solucionar y facilitar problemas, así bien de muchas opciones para crear estas soluciones. Existen muchas dimensiones de la misma, ya sea en función al tipo de interacción del usuario, al tipo de experiencia del usuario, a como este formado el propio sistema o, incluso, a si el procesamiento es en la red o local [10]. A la hora de desarrollar un generador de contenido procedural es de vital importancia tener en cuenta cuales son las cuestiones importantes para el sistema propio, para poder aplicar los métodos o algoritmos necesarios.

A su vez, la generación procedural de contenido en juegos (También conocida como PCG-G de "procedural content generation in games") es una herramienta muy frecuentemente utilizada para paliar la gran necesidad de generar contenido en un juego [2], pudiendo esta ser uno de los mayores cuellos de botella en el desarrollo de muchos juegos. De ahí que muchos juegos empleen técnicas como las nombradas previamente, como son los famosos juegos "Diablo 3", "No Man's Sky" o "Marvel's Spider-Man" para facilitar el desarrollo de los mismos.

No existe una solución única para cada problema, sino que se pueden aplicar infinidad de maneras para conseguir el objetivo previsto, pero si que resaltan algunas tendencias para algunos problemas más frecuentes.

## 2.1 Técnicas

Dentro del campo de la generación procedural, independientemente del objetivo a desarrollar, existen ciertas técnicas que son frecuentemente utilizadas para poder generar contenido de manera eficiente y ordenada.

### 2.1.1. Números pseudo-aleatorios

Los números pseudo-aleatorios son las base de cualquier clase de contenido generado proceduralmente. En lugar de tener un generador de números completamente aleatorios que cada vez dará un resultado diferente, los números pseudo-aleatorios se basan en la capacidad de ser reproducibles. Los números pseudo-aleatorios se generan mediante un algoritmo, que puede variar, y para cada semilla o número inicial, el resultado de este algoritmo será siempre el mis-

mo. Dada esta característica, un desarrollador puede reproducir el mismo patrón de números para un mismo experimento, donde se reflejaría solo los cambios en el propio código como causa de los diferentes resultados de los mismos [1]. Dada esta reproducibilidad, el contenido generado proceduralmente puede ser recreado simplemente con el conocimiento de la semilla propia. En la figura 2.1 se puede apreciar que los numeros generados son todos en función del primero.

$$
\begin{aligned}
\mathbf{Z}_1 &= f(\mathbf{A}, \mathbf{Z}_0) \\
\mathbf{Z}_2 &= f(\mathbf{A}, \mathbf{Z}_1) = f(\mathbf{A}, f(\mathbf{A}, \mathbf{Z}_0)) \\
\mathbf{Z}_3 &= f(\mathbf{A}, \mathbf{Z}_2) = f(\mathbf{A}, f(\mathbf{A}, \mathbf{Z}_1)) = f(\mathbf{A}, f(\mathbf{A}, f(\mathbf{A}, \mathbf{Z}_0))) \\
\vdots \quad & \vdots .
\end{aligned}
$$

**Figura 2.1:** Ejemplo de generador de números pseudoaleatorios, extraido de [1]

### 2.1.2. Ruido perlin

El ruido Perlin es una función matemática que resulta en un conjunto de valores pseudo aleatorios basados en la computación para cada punto $x, y, z$ del gradiente de los 8 vertices más cercanos de la red cúbica de enteros y luego haciendo interpolación entre estos valores. El resultado es un mapa de valores entre -1 y 1 para toda combinación de $x, y, z$ enteros[8]. Esta herramienta ve mucho uso en todo campo en referencia a gráficos al poder generar una secuencia de números pseudoaleatorios de manera repetible y eficiente computacionalmente. Es usada entre otros casos para la generación de texturas realistas de manera procedural. Las señales tienen la forma que se muestra en la figura 2.2.

Para generar estas señales de ruido se requerirá de modificar ciertos atributos de la señal: La frecuencia de la onda, la cantidad de octavas y la persistencia de la misma y la lagunaridad (o lacunarity en inglés). Estos valores representan lo siguiente:

- La frecuencia define la amplitud de la onda que genera la señal de ruido, siendo esta más amplia a valores más bajos.
- La cantidad de octavas representa la cantidad de señales superpuestas utilizadas
- La persistencia se refiere a la influencia que tiene cada una de las octavas en el resultado final
- La lagunaridad se refiere a el grado de heterogeneidad dado en la señal, a más alto más homogéneo

### 2.1.3. Gramáticas BNF

La notación Backus-Naur (BNF) es un metalenguaje utilizado para definir lenguajes formales y gramáticas libres de contexto. Una especificación de BNF[7] es un sistema de reglas descritos como

```
<simbolo> ::= <expresión con símbolos>
```

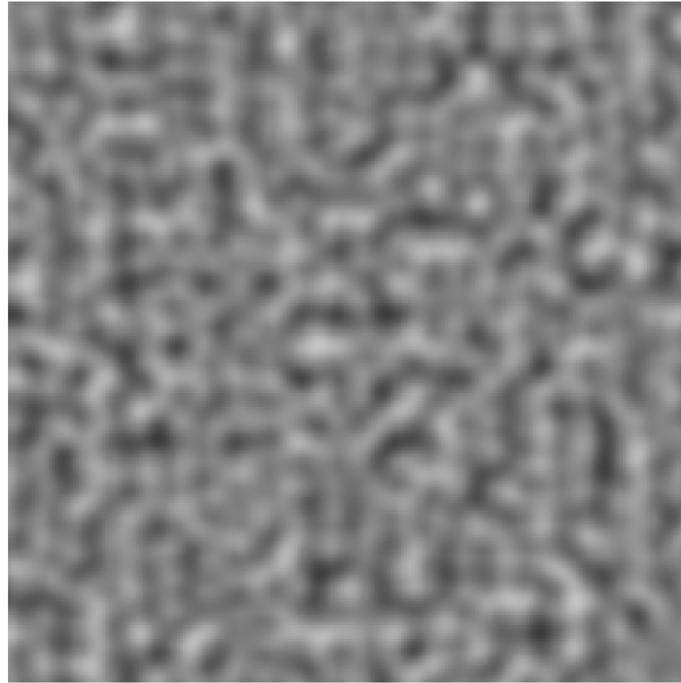

**Figura 2.2:** Señal de ruido de alta frecuencia, extraido de [8]

donde <símbolo>es un elemento no terminal mientras que la expresión es una secuencia de símbolos o secuencias separadas por "|", indicando que se produce una de ellas. Aquellos símbolos que no especifican otros elementos se denominan terminales.

Un ejemplo de una gramática escrita en notación BNF sería la siguiente, la cual provee una definición de una operación aritmética simple [5]:

```
<digit> = 0|1|2|3|4|5|6|7|8|9
<unsigned integer> := <digit> | <unsigned integer> <digit>
<adding operator> := +| -
<multiplying operator> := x | / | ÷
<primary> := <unsigned number>
  | <variable>
  | <function designator>
<factor> := <primary>
  | <factor> <primary>
<term> := <factor>
   | <term> <multiplying operator> <factor>
<simple arithmetic expression> := <term>
  | <adding operator> <term>
  | <simple arithmetic expression> <adding operator> <term>
```

Un resultado de esta gramática sería, por ejemplo, $12 + 33 - 1x2$

## 2.2 Terreno

Uno de los campos más usados de la generación procedural es en el ámbito de generación de terrenos y paisajes. Un punto importante antes de generar cualquier clase de terreno es centrarse en como de detallado queremos que sea este mismo, un simulador de vuelo no tiene necesidad de que el terreno sea excesivamente realista en su geometría si no vas a interactuar apenas con este, mientras una aplicación centrada en físicas puede tener interés de un terreno más definido. La razón por la que tenemos que tener en cuenta esta decisión es el uso de recursos y coste computacional, donde las formas más fáciles de generar terreno serán también las más simples y rápidas, aunque puede que el resultado no sea muy realista.

La manera más sencilla de generar un terreno de manera procedural es mediante el uso deseñales de ruido. El ruido, a su vez, en su forma más sencilla es una matriz de 2 dimensiones llena de valores pseudo-aleatorios. Dado esto, el terreno procedural más simple posible será aquel que para cada par de valores de $X, Y$, use como altura el valor del ruido asociado a esos valores. Esto se llama Heightmap y a pesar de su simplicidad se utiliza en videojuegos mundialmente conocidos como es Minecraft, aunque este presenta una variante del Heightmap basado en el mismo proceso con *Voxels* en 3D, dividiendo el espacio en cubos.

El problema del acercamiento anterior es el siguiente: si utilizamos un Heightmap completamente aleatorio el resultado serán un gran conjunto de valores muy diferentes unos de otros a pesar de la cercanía de los mismos. Por ello las siguientes técnicas que se utilizan para la generación se basan en la idea de que las montañas reales, a pesar de la existencia de acantilados [9], son en su mayor parte más suaves en sus cambios de altura. Teniendo esto en cuenta, se utiliza la generación de terrenos basada en la interpolación de valores. Esta funciona generando unos valores específicos del terreno y interpolando los valores intermedios. Esta interpolación puede ser bilinear, siguiendo a una función lineal para la interpolación, o bicúbica, basada en una ecuación simple de tercer grado como $s(x) = -2x^3 + 3x^2$ [9]. La siguiente técnica cada vez más realista se basa en el uso de gradientes. El ruido perlin es una versión de la idea del ruido basada en estos mismos gradientes, los cuales son interpolados entre los valores previamente mencionados. El resultado es un terreno más orgánico pero que fluctúa de manera constante y visible. El terreno real no presenta dichas fluctuaciones, sino que tiene diferencias a menor y mayor escala. La manera de poder representar estas diferentes escalas en una combinación es la idea en la que se basa el terreno fractal. El terreno fractal se puede generar de muchas maneras, pero la más común es la combinación de un mismo terreno, repetido diversas veces a diferentes escalas, multiplicando los valores de menores escalas por un factor decreciente, así influyendo en menor medida al resultado final.

Otros métodos existen para la generación de terrenos como son la generación basada en agentes, quienes generan distribuyendo tareas entre diferentes agentes como “conectores” que enlazan diferentes zonas de terreno o “extensores” que amplían zonas ya existentes, por poner un ejemplo. Por otro lado, también existen métodos de generación basados en búsquedas, como podría ser la programa-

ción de terreno genético (GTP), basada en sistemas evolutivos como autómatas celulares.

## 2.3 Laberintos y mapas

Uno de los grandes campos de la generación procedural usado en la actualidad es el enfoque en creación de laberintos, especialmente para el desarrollo de videojuegos. Refiriéndose a laberinto o mazmorra a un espacio interconectado entre si para la navegación por el mismo, existen diferentes formas de generarlos. El primer acercamiento más simple se basa en un acercamiento algorítmico para conseguir el objetivo, un algoritmo de partición del espacio. Los algoritmos de partición de espacio toman un determinado espacio ya sea en 2D o 3D y devuelven una serie de subdivisiones de los mismos tal que todo punto del espacio esté contenido en una de estas subdivisiones. Estos algoritmos típicamente operan de manera jerárquica, aplicando el algoritmo de manera recursiva [9]. El acercamiento más popular es la división del espacio de manera binaria (BSP), donde mediante estructuras de árboles binarios se divide el espacio en espacios más pequeños y determinando los enlaces entre estos mismos en referencia al propio árbol. El resultado de una ejecución de este algoritmo se puede ver en la figura 2.3.

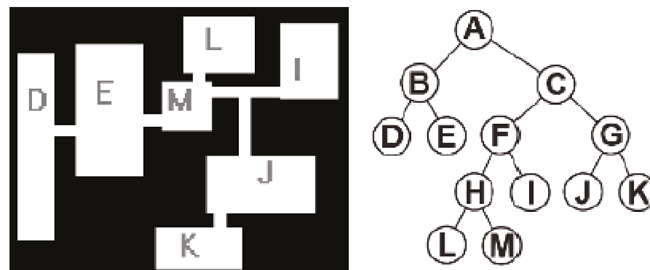


**Figura 2.3:** Resultado de ejecución de un algoritmo de partición del espacio, extraido de [9]

De la misma manera que en otros campos, también se pueden aplicar agentes a la hora de generar un laberinto. Estos métodos usualmente toman un agente que toma decisiones de manera estocástica sobre hacia que dirección continuar la generación. Dentro de esta definición general existen muchos cambios que pueden hacer que el agente se comporte de manera diferente, ya sea la probabilidad de cambiar de dirección, la capacidad de planificar a futuro en contra de una estrategia completamente aleatoria o la región de espacio cercano que tiene en cuenta a la hora de la toma de decisiones.

Por otra parte, métodos genéticos como son los autómatas celulares pueden ser una alternativa para conseguir otra solución a esta necesidad. Su funcionamiento es simple y similar a otros tipos de autómatas celulares, se define un espacio y una serie de reglas (en este caso para determinar terreno como accesible e inaccesible) y se distribuyen en el espacio células para terreno accesible e inaccesible. Tras ello se actualiza el autómata durante una serie de etapas para finalizar la

generación. El resultado de la generación de los autómatas celulares es un cuerpo cavernoso de apariencia relativamente orgánica, como en la figura 2.4.

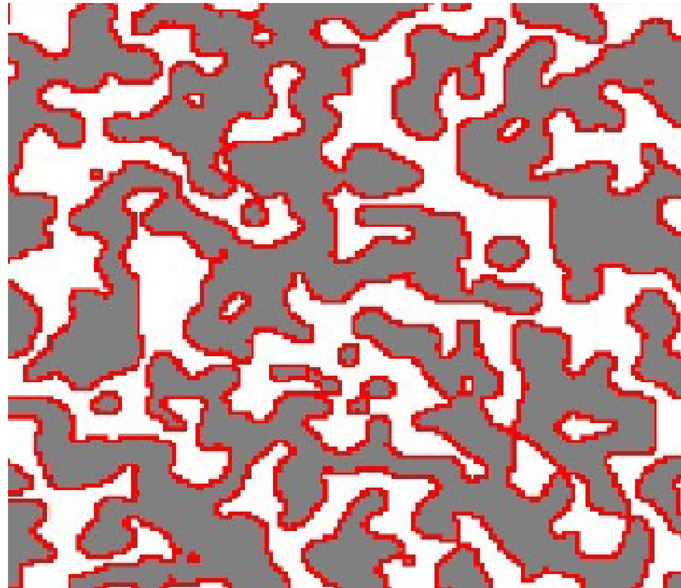

**Figura 2.4:** Resultado de ejecución de un autómata celular para generar un laberinto, extraido de [9]

Cada vez más, este campo es más innovativo y esto se presenta mediante la aplicación de las más novedosas tecnologías para resolver este mismo problema. Uno de estos acercamientos es el presentado en [4], proponiendo y desarrollando un generador de estas mazmorras basado en aprendizaje profundo, con el objetivo de crear los niveles de un juego. Si bien este acercamiento presenta nuevas posibilidades a la hora de generar contenido procedural, presenta nuevos desafíos a solventar, siendo estos el analizar el contenido generado desde el propio modelo que lo genera y la necesidad de continuar de pruebas humanas para experimentar el contenido generado.

## 2.4 Producciones gramaticales

Otro gran campo de la generación procedural es el de generación mediante gramáticas como las gramáticas BNF o "L-systems" para el objetivo de generar tanto producciones de forma orgánica como son plantas [9], comenzando con un espacio determinado e iterándolo o generando a partir de lo existente. Por otro lado, otras técnicas como "Split grammars" o "Shape grammars" han sido usadas con los mismos objetivos, si bien funcionando de maneras diferentes. Todos estos métodos se basan en la creación de las construcciones por partes definidas, si bien hay otras basadas en el uso de números pseudoaleatorios, centrándose en el diseño de planos del suelo y extendiendo estos planos verticalmente. Estas gramáticas y sistemas no están limitadas a ningún tipo de representación, ya sea 2D o 3D, sino que la propia representación debe ajustarse al problema u objetivo deseado. Un ejemplo es en el artículo de S.G.Mueller [6] donde define una gramática para la generación de una serie de construcciones en un ambiente urbano, siendo visible el resultado en la figura 2.5. Para ello la gramática deseada se di-

vide en diferentes niveles de prioridad donde se divide la problemática cada vez en problemas más pequeños, comenzando con un espacio donde estará, se divide en fachadas que además se dividen en “tiles” con sus propias entradas, ventanas, puertas y en nivel más pequeño los propios bloques y tejados.

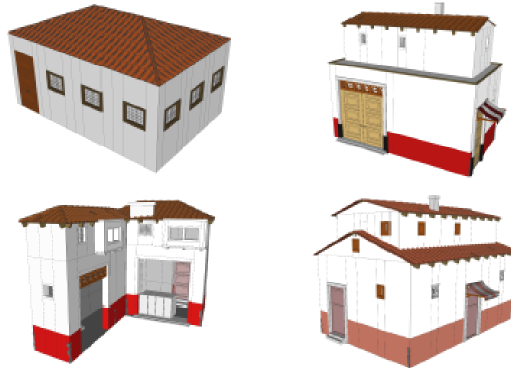

**Figura 2.5:** Edificios generados mediante diferentes gramáticas, extraido de [6]

Dentro de las producciones gramaticales, podemos incluir otros métodos como es la generación de laberintos mediante gramáticas, donde al contrario de los métodos anteriores, su aplicación no es para determinar el espacio disponible, sino el orden en el que los elementos deben estar ordenados para la correcta navegación en el mismo. Por ejemplo, como en la imagen 2.6 para abrir la puerta debes antes haber recogido la llave, lo cual sería imposible si esta estuviera tras esta misma puerta.

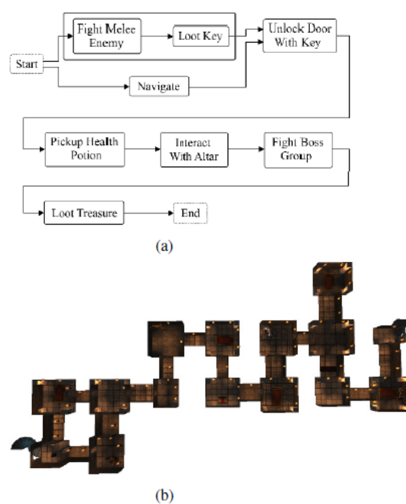


**Figura 2.6:** Ejemplo de mazmorra generada mediante una gramática, extraido de [9]

## 2.5 Otras producciones

Aparte de aquellos comentados previamente, existen muchos más tipos de generación procedural y aplicaciones de la misma. Uno de estos podría ser un acercamiento algorítmico a la cuestión de generación procedural. Se han desarrollado sistemas como el sistema *Citygen* [3], el cual genera en diferentes etapas una ciudad. Tomando este como ejemplo, el acercamiento algoritmo del mismo consiste en 3 pasos. El primer paso es generar las carreteras principales de la ciudad, creando un grafo de adyacencia para determinar diferentes puntos y cuales de ellos estarán conectados estre sí. El segundo paso es crear carreteras secundarias. Estas carreteras serán creadas en las zonas delimitadas por las divisiones principales. Finalmente, dividirá el espacio entre estas carreteras secundarias para crear edificios de diferentes dimensiones en estas divisiones. Este acercamiento es similar al algoritmo de división del espacio explicado antes, pero aplicado para un fin diferente.

Existen una infinidad de campos de generación procedural, ya sea animación procedural, generación procedural paramétrica, generación procedural de modelos o una larga lista de ellos, pero en este TFG se han presentado conceptos sobre unos cuantos de ellos.

## 2.6 Crítica al estado del arte

Existen infinidad de técnicas y formas de generar proceduralmente contenido, al ser la propia generación procedural un concepto general que abarca un gran abanico de posibilidades. Sin embargo, al tener disponibles tal cantidad de variaciones de como solucionar un mismo problema se presentan 2 problemas principales:

- No existe una forma única e infalible de solucionar un problema.
- No existe una herramienta única para generar contenido.

El primer punto es fácilmente explicable. Dependiendo del problema que se nos presente la solución necesitará ser más sencilla o complicada. El mejor resultado en diferentes casos como en el caso de uso para videojuegos es puramente subjetivo y no existe una métrica exacta determinante de cual es más adecuada. Por ejemplo, en un simulador de fluidos precipitándose sobre un terreno será preciso que el terreno sea lo más realista y detallado posible para poder reproducir todas las propiedades de los fluidos de manera certera. Por otro lado, en un videojuego de una consola, con espacio de memoria limitado, querremos que este contenido procedural siga existiendo, pero sea más ligero para acomodar las restricciones presentes.

El segundo punto se desencadena en base al primero. Al no existir una única solución para cada problema, cada uno de estos tiene que ser resuelto de manera específica. Y dentro de diversos generadores que funcionen sobre los mismos algoritmos pueden cambiar dependiendo de las necesidades de cada equipo de desarrollo. El mismo hecho de que cada equipo de desarrollo deba crear su propio

generador procedural adaptado única y exclusivamente a su problema resulta en que no existe actualmente una herramienta abierta a la comunidad para la generación procedural de diferentes cuestiones.

Otros problemas menores pero igualmente importantes surgen de los dos anteriores, como sobre qué plataforma está construido el generador, en qué formato se genera el resultado, qué lenguaje utiliza, etc. El campo de la generación procedural es uno muy personalizado y subjetivo dado la gran cantidad de posibilidades.

## 2.7 Propuesta

El campo de la generación procedural es uno muy extenso, abarcando todo tipo de problemas, soluciones y maneras de operar. Vistas todas estas posibilidades, este trabajo pretende no innovar sobre una nueva técnica, sino crear una herramienta nueva, de sencillo uso, que recoja varias de estas técnicas para la generación procedural. Este objetivo se plantea como un aporte a la comunidad de disponibilidad y facilidad para todos aquellos que deseen utilizar la generación procedural sin invertir una gran cantidad de tiempo en crearla ellos mismos.

# CAPÍTULO 3
# Decisión de una solución

Dada la propuesta previamente comentada es preciso acotar todas las posibilidades, comentadas y no comentadas, para poder elegir una en específico para desarrollar. Una selección sin antes haber valorado las posibilidades disponibles puede resultar en dificultades más adelante en el desarrollo o en tratar de desarrollar algo ya existente.

## 3.1 Identificación y análisis de soluciones posibles

Dentro de todas las posibilidades comentadas en el capítulo anterior, para completar la propuesta hecha debemos plantear la posibilidad de combinar varias de estas técnicas de manera coherente.

Una posibilidad es la de crear algun par de sistemas generativos y combinarlos. Un ejemplo sería un acercamiento algorítmico para generar , por ejemplo, plantas y otro generador basado en una gramática para generar otro tipo de elemento. A su vez, podría generarse esta misma planta y animarla proceduralmente basándose en un sistema de viento. Otra idea diferente, podría ser el combinar un generador de modelos para crear unos personajes y animarlos de manera procedural. Este es un acercamiento común en ciertos videojuegos como puede ser "Spore".

Por otro lado, cabe la posibilidad de generar un terreno de maneras diferentes. Es posible utilizar una técnica de generación de terreno procedural y luego manipularlo o juntarlo con otros mediante el resultado de un laberinto procedural, siendo estos terrenos cada una de las particiones del laberinto. Otro acercamiento con el terreno sería decorar el mismo, como podría ser utilizando una construcción gramatical para ello pudiendo, por ejemplo, crear un terreno como si este fuera un jardín y desarrollar un generador procedural de plantas y flores basado en una gramática, como descrito en [9].

Existen infinidad de posibilidades y sería posible combinar cualquier par de técnicas para crear una nueva herramienta.

## 3.2 Solución propuesta

Vistos todos los requisitos y posibilidades en apartados anteriores, la solución central en la que se basará este trabajo es la siguiente: crear una herramienta de generación procedural con dos partes, primero un generador de terreno fractal creado mediante ruido "perlin" y, por otro lado, un generador de edificios basado en una gramática. El resultado final deberá ser una librería mediante la cual se pueda generar un terreno basado en una semilla y sobre este generar construcciones en función de otra señal de ruido.

Con el objetivo de desarrollar la misma, primero se diseñará detalladamente tanto la arquitectura en la que se basa la librería como el diseño detallado de esta y como funcionan todas sus partes. En cuanto al desarrollo, primero se recogerán todas las librerías necesarias para el correcto desarrollo y se procederá, en primera instancia, a la creación del generador de terreno procedural.

Una vez desarrollado y comprobado el correcto funcionamiento, se procederá a la creación del generador de construcciones. Para ello primero se diseñará una gramática a seguir y en base a esta se hará. Una vez todo haya sido desarrollado y probado, se implantará en la forma de una demo que demuestre todas las posibilidades de la misma, siendo modificables diferentes parámetros para probar diferentes combinaciones.

Tras ello, la validación se basará en pruebas sobre esta propia demo, comprobaciones sobre la adecuación del terreno y las construcciones, adecuada posición de las construcciones sobre el terreno y verificación de aparición de formas de interés y de forma natural, más que un reparto aleatorio.

Finalmente se comprobará la capacidad de reproducibilidad del sistema sobre la misma demo. La reproducibilidad es un aspecto crucial y central de la generación procedural, ya que si fuera eliminada la misma del sistema, funcionaría de manera puramente aleatoria. Además, el objetivo de la solución es un generador accesible para todos los usuarios del mismo y que estos puedan de manera sencilla volver a generar resultados que otros obtuvieron anteriormente.

## 3.3 Plan de trabajo

Para poder abordar el trabajo necesario para completar este trabajo, será necesario ordenar el desarrollo de tal manera que queden claros objetivos y metas claras durante todo el tiempo que este ocupe.

Primeramente, se comenzará con el diseño y posterior desarrollo del generador de terreno, iterando en pasos pequeños desde uno simple hasta el generador final que se utilice. El objetivo es completar este paso dejando gran margen de tiempo para abordar el resto de apartados. Este apartado debería estar completado en alrededor de unas 50 horas, teniendo en cuenta la investigación inicial, el desarrollo y los posibles problemas surgidos en el mismo.

Después se procederá al diseño y desarrollo del generador de estructuras y, al igual que en el caso anterior, se desarrollará iterando en pasos pequeños y comprobando la funcionalidad añadida en cada uno de estos. Ha de tenerse en

cuenta de que previamente al desarrollo, en el diseño, se diseñará una gramática sobre la que se construirá este generador. El trabajo requerido para completar este apartade debería ser similar a la cantidad necesaria para el generador de terreno, unas 50 horas.

Una vez ambos generadores hayan sido desarrollados, se procederá a construir el generador combinado y la demo que lo complementa. Si bien el generador no debería ser tan intenso en cuanto al trabajo, la cantidad de pruebas que se deben hacer para confirmar el correcto funcionamiento harán que el tiempo estimado necesitado sea tanto como el de el resto de partes, 50 horas, o más incluso.

Es necesario destacar que durante todo el diseño y desarrollo de todas las partes anteriores se deberá apuntar y anotar todas aquellas cuestiones que surjan durante el desarrollo, para más tarde ser incluidas en esta propia memoria del trabajo.

La memoria será la parte individual más larga, teniendo que recopilar y explicar todas las cuestiones relevantes surgidas ya sea previamente o durante la realización del trabajo. Eso añadido a la investigación de trabajo de otros autores, la búsqueda y creación del material visual incluido en la misma y las pruebas del sistema hacen que se estime el tiempo necesario para completar esta parte en 150 horas o más, dado a la necesidad de revisar y cambiar partes de la misma si fuera necesario.

Con el objetivo de ordenar el desarrollo se creará un tablero de Trello con los diferentes campos a abordar en el diseño, desarrollo y escritura de este TFG, viéndose un estado avanzado de este en la figura 3.1. Este incluirá las diferentes cuestiones relacionadas con ambos generadores, la escritura de la memoria, la creación de la demo, etc.

Cada uno de los objetivos, como son el desarrollo de cada uno de los generadores, será dividido en pequeñas tareas de objetivo concreto, para así poder comprobar sin duda si fueron completadas. A medida que sean completadas o aparezcan más se actuará de manera acorde.

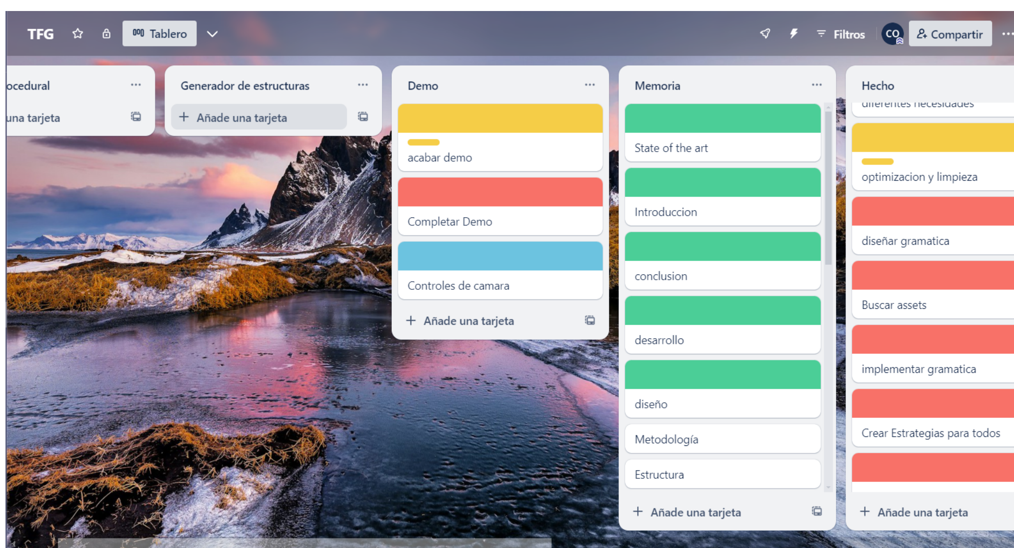


**Figura 3.1:** Captura de Trello con los diferentes campos en un estado avanzado del desarrollo

Continuando con los objetivos de organización, ya sabiendo todo el tiempo estimado para el desarrollo de todas las partes es necesario crear una previsión temporal para el desarrollo de todas las artes del trabajo. Primeramente, ya decidida la solución propuesta, se pretenderá realizar el desarrollo del generador de terreno procedural para tenerlo finalizado el dia 29 de marzo. Desde el comenzamiento del proyecto esto sería un tiempo de alrededor de un mes. El siguiente objetivo será completar el generador de construcciones para el día 24 de abril, otro mes para esta parte. Para el generador combinado, se pretenderá finalizar el desarrollo para el día 24 de marzo, pretendiendo ocupar la siguiente semana para el correcto funcionamiento de la demo y hacer diferentes pruebas. Finalmente, la memoria se irá escribiendo a lo largo que se vayan desarrollando las diferentes partes, teniendo como fecha final de entrega el dia 28 de junio, pero pretendiéndose finalizarla como muy tarde una semana antes, a día 21 de junio. Las fechas son visibles y señaladas en la figura 3.2.

marzo

| L | M | X | J | V | S | D |
|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

abril

| L | M | X | J | V | S | D |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | | | | | |

mayo

| L | M | X | J | V | S | D |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | | |

junio

| L | M | X | J | V | S | D |
|---|---|---|---|---|---|---|
| | | | | | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |

**Figura 3.2:** Calendario organizativo del desarrollo del TFG

# CAPÍTULO 4
# Diseño de la solución

Llegado el momento de diseñar la propia solución para crear ambos generadores, de terreno y construcciones, es necesario desglosar cada uno de estos en sus respectivas partes para poder comprender adecuadamente como organizar el desarrollo y mantenerlo ordenado.

## 4.1 Arquitectura del sistema

Primeramente, el generador de construcciones se dividirá en 2 grandes partes. La primera parte será un generador de objetos del tipo construcción y seguidamente tendremos un renderizador de estructuras. Además de estos, se creará un objeto "Opciones de construcción" el cual contendrá toda la información necesaria de como se tiene que crear el edificio. El funcionamiento será de la siguiente manera: Cuando se le pida al generador operar, este creará un objeto según los parámetros establecidos en las opciones. Una vez creado el objeto, este será pasado al renderizador, el cual lo renderizará en el espacio y devolverá una referencia a el mismo para posibilitar su futura manipulación. No existirá como tal un componente "Generador de construcciones", sino que será la unión de estos 2 explicados previamente.

Por otro lado, el generador de terrenos será un único bloque que contendrá toda la funcionalidad del diseño de terrenos. Esta funcionalidad incluye todos los parámetros necesarios al igual que otro objeto "Opciones de terreno", el cual contiene las opciones modificables del terreno. El funcionamiento será autocontenido, se llamará al generador y este mismo devolverá el resultado esperado. Cabe destacar que entre los parámetros propios de este se incluirán diferentes señales de ruido.

Para la combinación de ambos generadores se creará un nuevo bloque, conteniendo una instancia de cada generador y otro que una la funcionalidad de estos. Cuando este opere, primero utilizará su generador de terreno interno para generar el mismo y, seguidamente, detectará aquellos lugares donde deban ser generadas construcciones, llamando al generador de construcciones para este mismo objetivo.

Finalmente, otro bloque que incluirá la demo interactiva interactuará solo con el generador combinado, dándole a este los parámetros necesarios en la forma

de las opciones anteriormente comentadas, creando una interfaz para modificar algunos de estos valores de manera gráfica e interactiva. En definitiva, la arquitectura presentada tendrá la forma indicada en la figura 4.1.

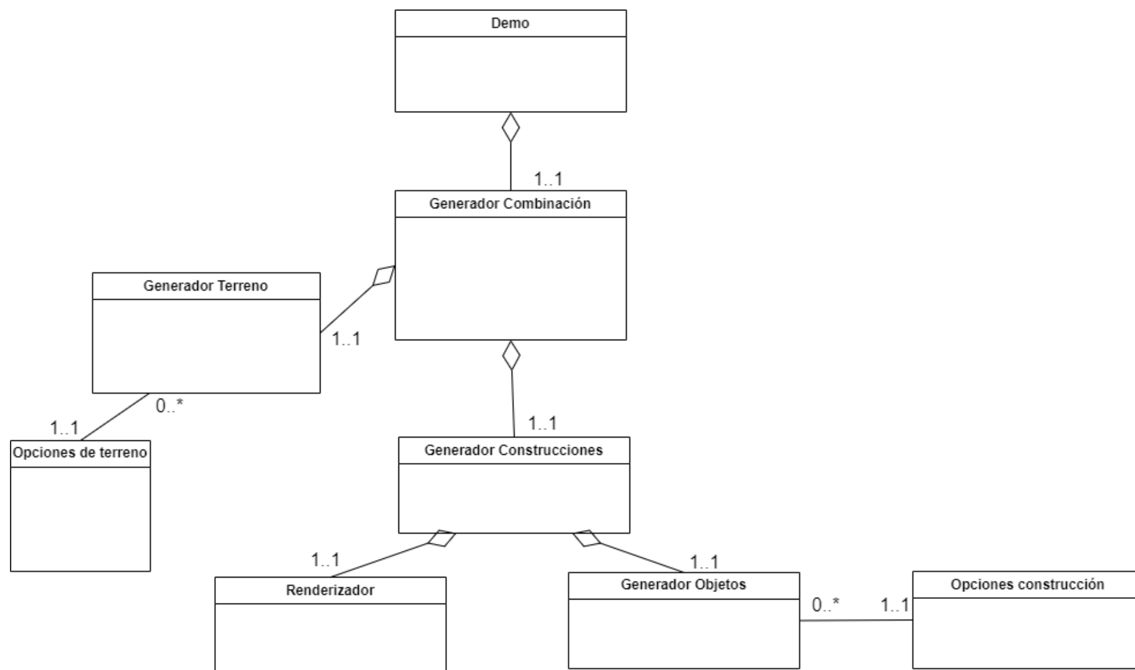


**Figura 4.1:** Diagrama UML de la arquitectura propuesta

## 4.2 Diseño Detallado

Centrándose más en profundidad en el diseño, podemos declarar las clases que tendrá el proyecto y la funcionalidad de cada una de ellas, así como la organización y relación entre ellas.

### 4.2.1. Terreno

Como ya fue comentado anteriormente, el terreno será generado mediante una clase creadora y otra clase que contenga los parámetros necesarios para esta funcionalidad, siendo esta la clase de opciones del terreno. Cabe resaltar que el objetivo es crear un terreno fractal mediante la combinación de diferentes señales de ruido, dato que será tomado en cuenta en todo el proceso. Para crear una malla en 3D necesitamos lo siguiente: una lista de vértices y otra lista de sus respectivos índices y un material que aplicarle a esta malla para la renderización. Para completar este proceso en el generador, las opciones deberán contener un material y unas dimensiones en X e Y. A parte de esto también contendrá una semilla en forma de entero para usarla como punto de inicio en el proceso procedural. Con las opciones definidas, el proceso que seguirá el generador será el siguiente:

1. Primero, recoger todos los valores de las opciones asignadas al mismo, para facilitar el posterior uso de los mismos.

2. Seguidamente, se inicializarán todas las variables y señales de ruido necesarias para el proceso generativo, siendo algunas de estas las listas de vértices e índices o el objeto malla.

3. A continuación, se pasará a crear los vértices de manera iterativa, tomando como parámetros para la señal de ruido la posición (X,Y) del vértice a generar. En esta fase se proveerá un valor opción para determinar de qué señal de ruido se pretende recoger el valor, ya sea cualquiera de las existentes o la combinación de las mismas. Dado que el valor recogido de la señal de ruido será entre 1 y -1, se precisará de un tratamiento de estos valores para mejorar la posterior visualización, consistente en una interpolación lineal entre estos valores y devolverá un valor entre 0 y un máximo modificable establecido en el generador.

4. Finalmente, se completará la creación de la malla con la declaración de los índices de cada uno de los triángulos que forman el terreno, así como la aplicación del material a la malla para su correcta visualización.

Cabe destacar que, para mejorar la visualización del terreno y sus detalles, se diseñará un material que cree un gradiente de colores dependiendo de la altura del punto en el espacio como se muestra en la figura 4.2, para así poder diferenciar a simple vista aquellos puntos de mayor altitud de otros más bajos.

A su vez, las señales de ruido que se seleccionen para la creación del terreno deberán ser distinguidas entre ellas, variando en tamaño de las ondas y nivel de suavidad de las mismas, para finalmente producir un resultado más realista.

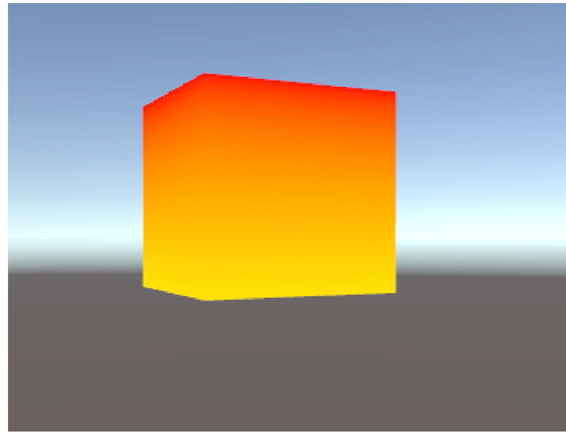

**Figura 4.2:** Ejemplo del efecto deseado con el material gradiente

Por motivos de organización, todo elemento relacionado con el propio generador de terreno será incluido en la carpeta */Generation/Terrain*, eso es la clase del generador, la clase de opciones y un objeto de esta misma clase al menos y el material del gradiente. Adicionalmente, se incluirá una sencilla demo para poder comprobar el correcto funcionamiento de este generador.

### 4.2.2. Construcciones

En cuanto al generador de construcciones se refiere, es necesario abordar 2 dimensiones, el generador y el renderizador. Sin embargo, antes de comenzar a diseñar ninguno de estos será necesario definir la gramática sobre la que se trabajará, puesto que el diseño de los generadores se basará en esta.

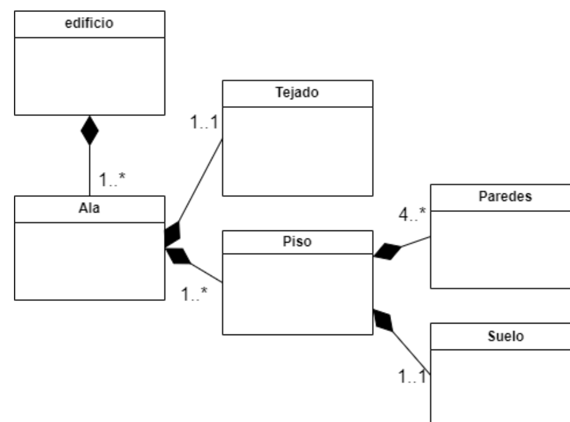


**Figura 4.3:** Ejemplo de diagrama UML para representar la estructura de un edificio

Sobre la gramática, se debe primero analizar cuales son los elementos necesarios para la creación de un edificio. Todo edificio tiene como mínimo un tejado, unas paredes y un suelo. Más allá de la definición mínima de edificio se plantean diferentes posibilidades, como podrían ser tener más de un piso o tener diferentes alas en un mismo edificio, cada una diferente. Una vez identificado los diferentes elementos de esta, la gramática puede ser diseñada, siguiendo la notación BNF con el objetivo de estandarizar el proceso. Un ejemplo de edificio es el que se presenta en la figura 4.3.

En concreto, para la gramática que definirá las construcciones se definirá lo siguiente: Un edificio que podrá ser dividido en diferentes alas. Cada una de estas alas contendrá un número de pisos mayor o igual que uno y un tejado entre los dos disponibles. Cada uno de estos pisos a su vez será compuesto por un suelo, que será igual para todos ellos, y una cantidad de paredes que lo envolverán. Según la gramática esto serán de 1 a infinitas paredes, por lo que en el desarrollo será necesario acotar este número, dejando la representación en la gramática como se presenta. Las paredes podrán ser de diferentes tipos, siendo aquellas que contengan en su nombre "W"quienes tendrán una ventana y las que tengan "D"tendrán una puerta. Por lo tanto la gramática definida será la siguiente:

```
<Building> := <Wings>
<Wings> := <Wing> | <Wing> <Wings>
<Wing> := <Levels> <Roof>
<Levels> := <Level> | <Level> <Levels>
<Roof> := RoofA | RoofB
<Level> := Floor <Walls>
<Walls> := <Wall> | <Wall> <Walls>
<Wall> := Wall1 | Wall2 | WallW | WallWW | WallD | WallDW
```

Con la gramática y los elementos de la misma definidos, se crearán las clases que constituirán las construcciones. Habrá una clase para cada uno de los elementos, estos siendo construcción, ala, piso, tejado y pared. Cada una de estas clases contendrá los parámetros necesarios para relacionarlos con el resto en función de la definición de la gramática y un constructor para completar el funcionamiento. Un ejemplo de esto sería que la clase construcción albergará parámetros para las dimensiones del mismo y las alas que la constituyen que, a su vez, contendrán una serie de pisos y tejado cada una.

Una vez definidas las clases, será preciso diseñar el generador de construcciones siguiendo la definición de la gramática. Para ello se tomarán todas las reglas definidas en esta y se interpretarán para crear el correcto ordenamiento de órdenes en el generador. Aquellos elementos que definan una secuencia infinita de los mismos serán ajustados dentro de un rango para no crear ninguna clase de ejecución inacabable. El resultado de este generador será un objeto de la clase construcción.

Por otro lado, el renderizador de estructuras tomará una construcción y siguiendo el mismo proceso que el generador, renderizará cada uno de los elementos. Cabe resaltar que no todos los elementos son renderizables, solo aquellos que sean elementos terminales en la gramática. Desde la construcción dada, se tomará sus alas, cada una con sus pisos y así, desglosándola, se renderizará cada elemento. Dentro del renderizador será preciso calibrar los valores de distancia entre elementos, así como la rotación de los mismos para la correcta visualización de estos. Una vez finalizado el proceso se devolverá una referencia al edificio creado.

Para el funcionamiento del renderizador, requerirá de una serie de assets representativos de aquellos elementos no terminales de la gramática, pues serán estos assets aquello que se renderice en el espacio de la aplicación. Estos serán paredes, suelos y tejados, como se muestran unos en la figura 4.4, pero serán más detallados en una sección futura.

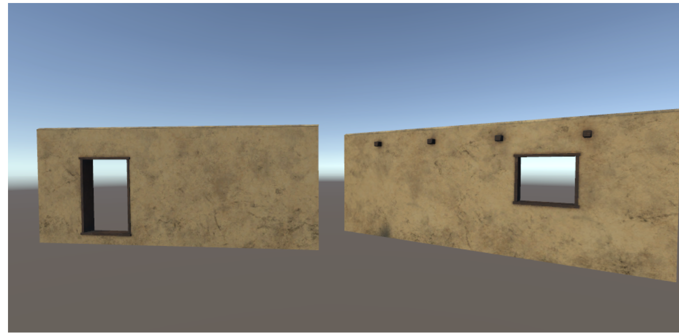

**Figura 4.4:** Ejemplo de assets utilizados para la generación de construcciones

Para alcanzar un funcionamiento modular, se creará la clase "opciones de construcción", la cual contendrá las estrategias para la generación de cada elemento. Las estrategias serán otras clases que contengan un método cada una para la generación de un tipo de elemento. El generador de construcciones utilizará un conjunto de estrategias predeterminadas a menos que en estas opciones se provea alguna estrategia diferente. Por lo tanto, se deberá desarrollar una estrategia predeterminada para cada elemento, así como alguna estrategia diferente en alguno de los campos para demostrar el funcionamiento de las mismas. Cada una de estas estrategias, en su método de generación, también deberá llamar al siguiente método generativo de otra estrategia o del propio generador.

Por motivos de organización, todo lo relacionado con la generación de construcciones será almacenado en /Generation/Buildings, donde estarán el renderizador de construcciones, el generador y las opciones. Además, en el directorio /Generation/Buildings/Classes se almacenarán todas las clases y en /Generation/Buildings/Strategies se almacenarán una carpeta por elemento que requiera una estrategia. Cada una de estas carpetas contendrá una clase abstracta de estrategia, una clase de estrategia predeterminada, que herede de la abstracta, y un objeto instancia de esta.

### 4.2.3. Generador combinado

Ya definidos ambos generadores, es preciso definir como será la combinación de ambos. El generador combinado será en esencia una única clase que interactúe con tanto el generador de terrenos como el de construcciones. Para conseguirlo esta clase heredará del generador de terreno y hará referencia a todas las funcionalidades necesarias del generador de construcciones. De esta manera este podrá acceder a todos los atributos del generador de terreno de forma sencilla, mientras que del generador de construcciones solamente es necesario poder generar nuevas estructuras.

El funcionamiento de esta clase será similar al generador de terrenos, cambiando el método de generar terreno donde se realizarán funcionalidades extra. Se generará una señal de ruido extra para la representación de estructuras, donde los puntos donde haya altos valores, mayores que un límite inferior, se usarán como lugar para generar construcciones. Una vez se encuentre un punto que cumpla las condiciones, se llamará al generador de construcciones para que genere un objeto y después al renderizador para que lo coloque en la escena. El generador

tendrá que tener en cuenta cuestiones específicas como son no generar construcciones el espacios ya ocupados, tener en cuenta un límite de la altura para no generar estructuras que parezcan flotantes o ajustar la posición en el eje Y de la misma dependiendo de la pendiente del terreno donde se sitúe.

En cuanto a organización, dado que la funcionalidad es similar a la del generador de terreno, este generador combinado se almacenará en /Generation/Terrain, siendo un solo archivo; la clase.

### 4.2.4. Demo

Finalmente, el diseño de la demo también deberá ser definido claramente. Esta demo pretenderá mostrar todas las capacidades posibles de ambos generadores dados unos parámetros modificables por el usuario mediante una interfaz interactiva. Más en concreto, deberá contener formas de modificar los valores más relevantes a la hora de generar tanto el terreno como las construcciones.

La demo contendrá tanto unas opciones de terreno como de construcciones asignadas ya previamente, las cuales no se podrán cambiar. Estas opciones inmutables deberán ser probadas para imponer unas que resalten las capacidades de ambos generadores. Sin embargo, deberá incluir la capacidad de modificar, como mínimo, los valores de:

- El tipo de terreno a generar.
- Los pesos de la operación del terreno.
- La semilla de los generadores.

Es posible también la inclusión de otros valores a modificar específicos aparecidos en el desarrollo. Dadas todas las funcionalidades deseadas, la demo debería tener la forma descrita en la figura 4.5.

Esta demo también deberá contener alguna manera de cambiar la perspectiva, rotando el terreno y acercando la cámara para así poder apreciar más de cerca las facciones del mismo y las construcciones.

## 4.3 Tecnología utilizada

Una vez ya diseñado todo el sistema, será necesario decidir qué tecnologias se utilizarán para las diferentes partes, ya sea para las construcciones o para el terreno.

### 4.3.1. Requisitos

Un aspecto muy importante a definir antes de comenzar el desarrollo del trabajo es toda la tecnología sobre la cual se va a realizar. Toda propuesta cercana a la programación gráfica, como requieren ambos el terreno y las construcciones, requiere de diversas herramientas específicas para su correcto desarrollo. Para la

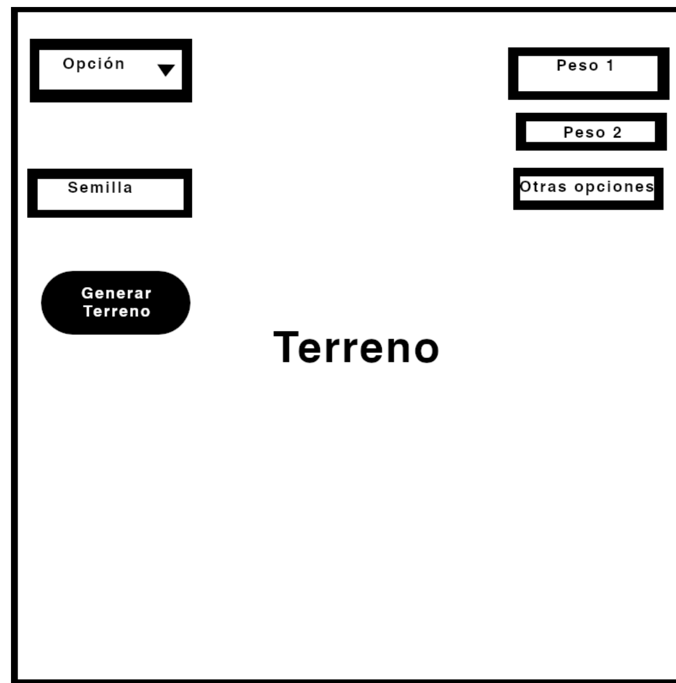


**Figura 4.5:** Diagrama mostrando la configuración prevista de la demo

mayor parte de lenguajes de programación existe alguna de estas, como podría ser OpenGl o DirectX. Por otro lado, existen diferentes motores gráficos que, aunque usualmente son utilizados para el desarrollo de videojuegos, también pueden ser utilizados con otros fines. Estos podrían ser Unity o Unreal Engine. A su vez, es necesario tener disponible alguna herramienta o librería que permita la generación de señales de ruido que más tarde serán utilizadas. La generación de señales de ruido no es un campo que pretenda cubrir este trabajo y es una cuestión suficientemente complicada como para ser necesario la incorporación de una librería externa. Finalmente, también serán necesarios por una parte un conjunto de assets o modelos de las diferentes partes de las construcciones, así como el material del gradiente, ambos comentados en la sección anterior. De estos existen una gran cantidad de manera gratuita así que la selección será en base a cuál cumple mejor las necesidades del trabajo.

### 4.3.2. Elecciones

Dados los requisitos y objetivos del trabajo, las herramientas utilizadas para el desarrollo del trabajo serán las siguientes. En cuanto a entorno de desarrollo, el motor gráfico de Unity será utilizado por diversas razones:

- Facilidad de visualización de gráficos
- Simplicidad de la programación gráfica
- Acceso a herramientas adicionales

Unity consiste de una serie de ventanas interactuables donde algunas de ellas te permiten visualizar en tiempo real el entorno, incluso realizar cambios en las

variables en el momento, como es posible ver en la figura 4.6. Además, el acercamiento a la programación gráfica de unity es uno simplificado, de manera que no tienes que realizar todos los pasos y algunos pueden ser realizados en los menús disponible, facilitando el desarrollo del código.

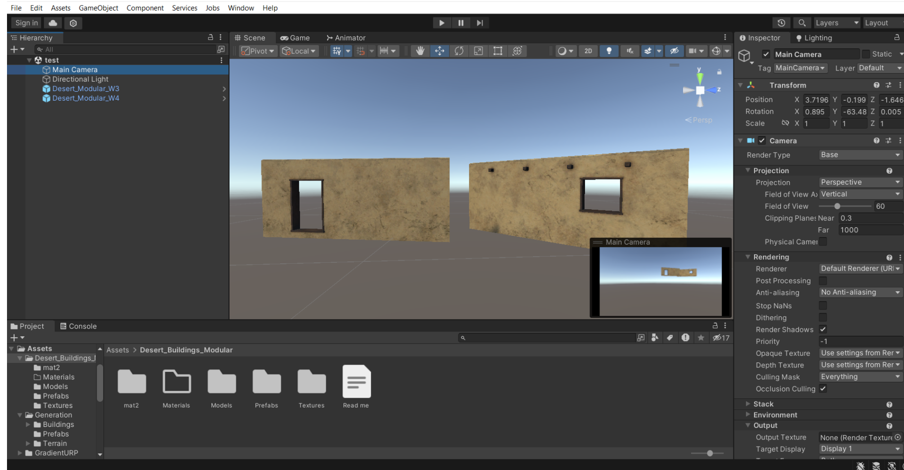


**Figura 4.6:** Entorno de Unity

Para la generación de señales de ruido en unity existe una versión adaptada de la librería LibNoise de C[1], y es esta adaptación la que será utilizada durante el desarrollo del proyecto. Esta contiene todas las funcionalidades y estructuras de datos necesarias para conseguir el objetivo deseado.

También cabe destacar que existen gran cantidad de herramientas extras creadas por la comunidad que pueden ser añadidas al proyecto, como aquellas descargadas desde la Unity Asset Store. De aquí es de donde se descargará el paquete gratis de construcciones modulares, por Lukas Bobor[2], visible en la figura 4.7.

**Figura 4.7:** Assets del paquete de construcciones por Lukas Bobor

[1]página de la librería: https://github.com/ricardojmendez/LibNoise.Unity

[2]Extraido de https://assetstore.unity.com/packages/3d/environments/urban/desert-buildings-modular-144178

Para el terreno durante el desarrollo se encontró la necesidad de un material que haga un gradiente de color con luces, sombras y reflejos y por lo tanto se tuvo que descargar uno de la Unity Asset Store. El seleccionado es [URP] Gradient Shader por Basic Stuff [3], como se puede apreciar en 4.8.

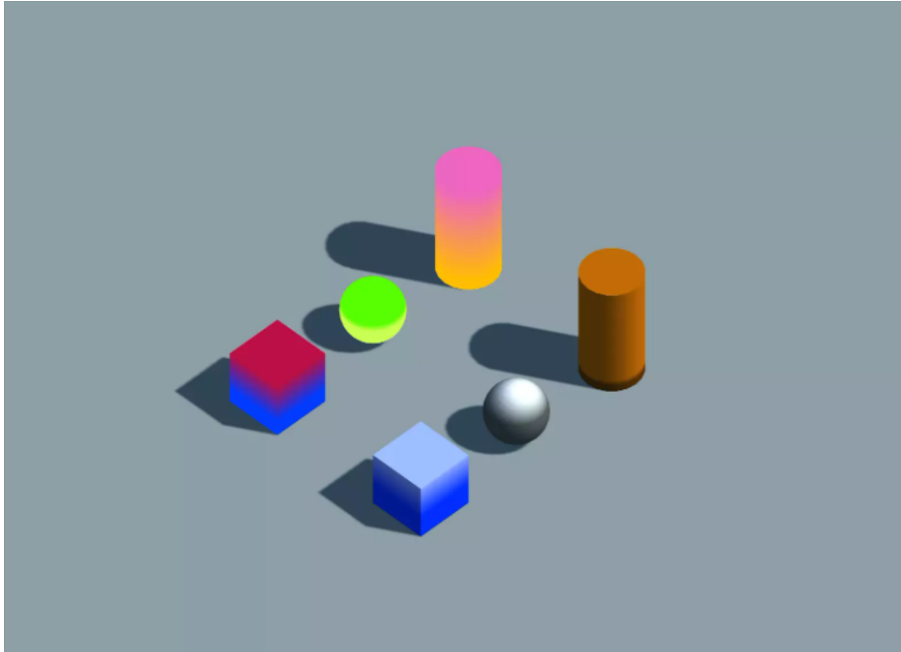

**Figura 4.8:** Materiales del paquete de gradientes por Basic Stuff

[3]Extraido de https://assetstore.unity.com/packages/vfx/shaders/urp-gradient-shader-lit-unlit-197492

# CAPÍTULO 5
# Desarrollo de la solución propuesta

El desarrollo de la herramienta de generación procedural ha experimentado muchas diferentes fases, cada una con sus problemas y soluciones que en esta sección se comentarán.

## 5.1 Preparación

Dado que ya fue definido que el proyecto se realizará en el motor de Unity, el primer paso es adecuar el entorno de trabajo para las necesidades del mismo. El primer paso para comenzar a trabajar con Unity es siempre crear un proyecto de Unity. Existen diferentes plantillas para proyectos de Unity, pero de entre ellas se seleccionará "Universal 3D" o un proyecto 3D utilizando la "Universal render pipeline", como se ve en la figura 5.1 . La razón de seleccionar este en concreto

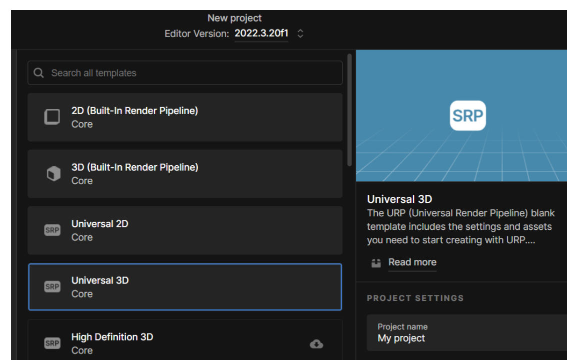


**Figura 5.1:** Ejemplo del menú de creación de proyecto de Unity

se basa en que este procesamiento de renderizado permite al usuario acceder a todas las funcionalidades de la programación gráfica. Una vez creado el proyecto deberá ser creada la jerarquía de carpetas para ordenar correctamente el proyecto. Las principales de estas serán las comentadas anteriormente: la carpeta *Generation* que contendrá *Buildings* y *Terrain*, donde se almacenará todo lo relacionado al generador de construcciones y terreno respectivamente. Dado el funcionamiento de Unity, donde las clases de código se organizan en *scripts* que deben enlazarse a un objeto para ejecutarse, deberemos crear un *GameObject* para que almacene la funcionalidad de los generadores.

Una vez esto ha sido completado, se deberá descargar y extraer en el proyecto tanto el paquete de *assets* como la librería del ruido. La librería *LibNoise* viene en formato zip, así que simplemente se necesitará colocar en la carpeta *Assets* (la carpeta raíz del proyecto) y descomprimir el archivo, generando una carpeta llamada *LibNoise.Unity-master*, que contiene todas las funcionalidades de la librería.

Por otro lado, este concreto paquete de objetos no está correctamente preparado para ser utilizado directamente, apreciable en 5.2, todos los objetos no tienen la textura asignada y por ello aparecen de color rosa, en lugar del color que aparece en las imágenes anteriores. Para solucionar esto deberemos crear un material para cada objeto, asignándole a cada material sus mapas de color y normales correspondientes. Una vez hecho esto, se le asigna el material creado al objeto y se podrá comprobar su correcto funcionamiento.

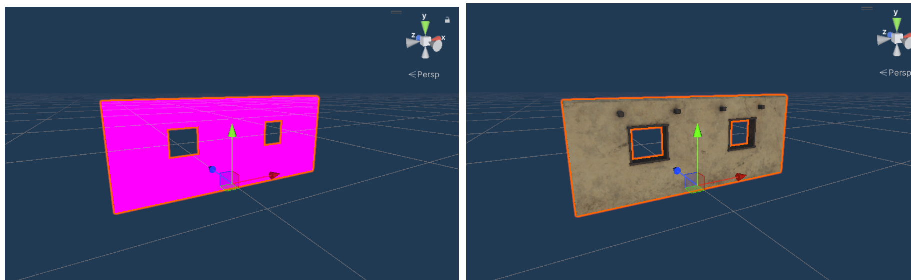

**Figura 5.2:** Pared del edificio como viene en el paquete y con la textura asignada

Una vez todos estos pasos han sido completados el desarrollo de ambos generadores puede ser comenzado.

## 5.2 Generador de terreno

El proceso de desarrollo del generador del terreno consistió en muchas fases las cuales construían sobre el trabajo anterior, pero para poder comenzar ha de conocerse como trabajar con la programación gráfica en unity. La manera más sencilla de trabajar con gráficos en unity es mediante el uso de objetos Mesh, aunque también existe la alternativa de trabajar con directivas de OpenGL.

Los objetos mesh contienen todos los atributos básicos necesarios para dibujar un objeto, siendo estos una lista de vértices e índices y, adicionalmente, otros parámetros como la lista de normales y colores de los vértices. La lista de vértices será cada una de las posiciones donde se definirán estos para luego crear los triángulos de la malla en base a los mismos. Para la creación de los índices (o triangles que se llaman en unity) se deberá conocer una característica propia de unity, debemos definir los índices de los triángulos en orden de las agujas del reloj. Por ello, si se desea crear una malla de mayor tamaño mediante un proceso iterativo, será importante tener este dato en cuenta.

Por tanto, para crear una malla en unity se proveerá una lista de vértices con un vértice definido para cada valor entero posible en el espacio definido en (X, Z), hasta un máximo definido por el usuario. El objetivo será crear una malla tal y como muestra el diagrama 5.3, haciendo los triángulos entre los vértices de la

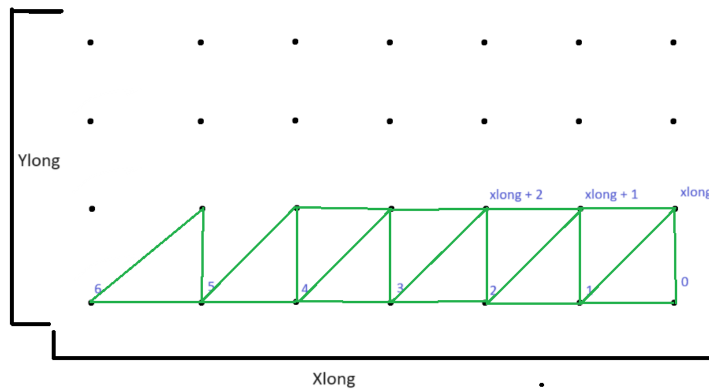


**Figura 5.3:** Diagrama explicando el funcionamiento del algoritmo que define los triángulos de la malla

fila actual y la siguiente. Los índices para cada triángulo se definirán como los siguientes:

```
Índices(t) = [t , t+1 , t+xlong  , t+xlong+1 , t+xlong , t+1]
```

para el triángulo nº t.

La ordenación de los índices de esta manera se debe al orden en el que se definen los triángulos en el espacio, por un método iterativo que atraviesa primero filas y luego columnas, siendo *xlong* la longitud de una fila. Siguiendo este algoritmo será necesario saltar esta definición de triángulos cuando se encuentre en el ultimo elemento de cada fila. Por tanto, el método encargado será el siguiente:

```
void CreateIndices(int xlong, int ylong){
    int cumm  = 0;
    int basic = 0;
    for(int i = 0; i < xlong-1; i++){
        for(int j = 0; j<ylong; j++){
            if(j == ylong-1){
                basic++;
                continue;
            }
            triangles[cumm++] = basic;
            triangles[cumm++] = basic+1;
            triangles[cumm++] = basic+xlong;
            triangles[cumm++] = basic+xlong+1;
            triangles[cumm++] = basic+xlong;
            triangles[cumm++] = basic+1;
            basic += 1;
        }
    }
}
```

El siguiente paso para generar el terreno es definir las señales de ruido que serán utilizadas. Para el generador de terreno que se pretende desarrollar se crearán 3 señales de ruido diferentes con distintos niveles de detalle:

- Una primera señal con ondas grandes y suave
- Una segunda señal igualmente suave, pero con ondas más pequeñas y detalladas
- Una tercera señal con ondas grandes, pero de aspecto rugoso y detallado

Los valores probados que generan este tipo de señales serán:

| | Frecuencia | Lagunaridad | Número de octavas | Persistencia |
|---|---|---|---|---|
| Noise 1 | 0.02 | 2 | 3 | 0.3 |
| Noise 2 | 0.06 | 2 | 3 | 0.1 |
| Noise 3 | 0.04 | 2 | 5 | 0.5 |

Una vez ya definidas las señales de ruido, se conseguirá generar el terreno deseado, tomando los valores de las señales de ruido para la altura. Sin embargo, si se fuera a hacer eso, el terreno aparecería sin textura y si se le añadiera un material de color plano, todo el terreno sería del mismo color, no dejando apreciar las diferencias en la altura ni los detalles del terreno. Es por eso que para poder apreciar el terreno más adecuadamente, un material nuevo será desarrollado, coloreando este el terreno dependiendo de la altura del mismo. Para ello se crea un archivo shader donde se tomará la altura del punto y se hará una interpolación lineal entre ambos. El valor de la interpolación lineal vendrá dado por la operación:

$$L = vertice.y / MaximoAltura \tag{5.1}$$

Este valor estará comprendido entre 0 y 1 y representará el porcentaje de cada color en la interpolación.

Ya con todos los componentes necesarios para la creación del terreno se procedió a hacer una serie de primeros acercamientos. El primero y más simple consistió en la comprobación del correcto funcionamiento de la librería generadora de las señales de ruido, de tal manera que se generara correctamente la señal de ruido y se pudieran acceder los valores tal y como se esperaba. El resultado fue un terreno formado de cubos predeterminados del motor, donde en el lugar que ocuparía cada vértice se hallaba uno de estos, como se puede apreciar en 5.4. Dado que este acercamiento solo tenía el propósito de comprobar el correcto funcionamiento de todos los componentes, se abandonó después de dicha comprobación.

Ya preparado todo para una primera iteración del terreno con la malla, se procedió a primero hacer una primera prueba para comprobar el funcionamiento y, después, comprobar los límites de los parámetros del generador. Diversas conclusiones surgen de esta primera prueba:

Primeramente, el generador no puede generar terrenos de tamaño mayor que 300x300 en una misma llamada, al no poder computar en 1 frame (o 1 actualización del motor) todo el algoritmo tantas veces, visible en 5.5. Esto se debe a la

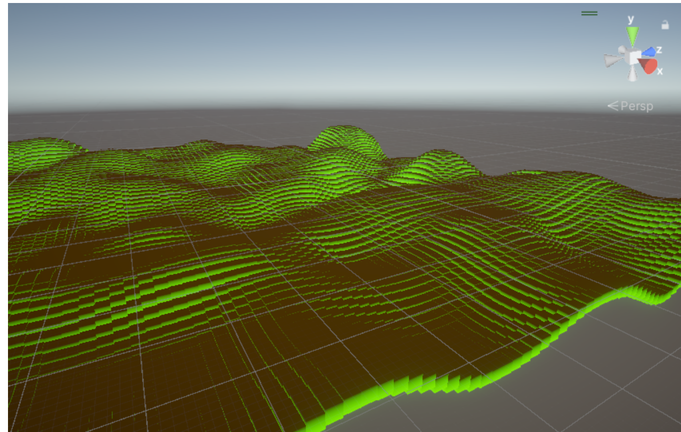

**Figura 5.4:** Terreno generado con cubos para comprobar el funcionamiento de la librería

complejidad del mismo, siendo de $n^2$ y que la cantidad de iteraciones a realizar con estas dimensiones sea más del doble de las que se iterarán con las dimensiones con que se mostrará el terreno; 200x200, también visible en 5.5.

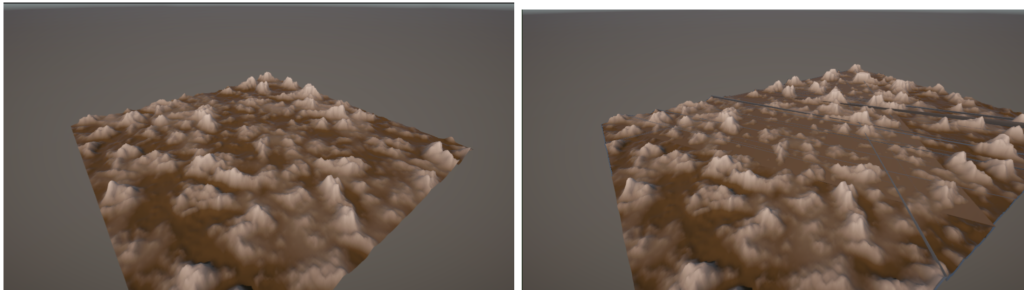

**Figura 5.5:** Terreno de 200x200 correctamente generado a la izquierda. Terreno de 300x300 incorrectamente generado a la derecha

Otra cuestión surgida de esta primera prueba es que las normales de los puntos no están definidas por defecto ni son definidas mediante el método de generar los vértices. Esta cuestión es de fácil arreglo, los objetos mesh tienen un método llamado *RecalculateNormals*, el cual calcula las normales a partir de los vértices. Para evitar problemas futuros también se añadirán los métodos *Optimize* y *RecalculateBounds*, que eliminan datos repetidos y establecen el espacio que ocupa la malla respectivamente.

Otro de estos problemas es que Unity no reconoce bien el material a aplicar al terreno si este es aplicado desde la interfaz gráfica. Este problema es de fácil solución, simplemente aplicarlo desde el código lo solucionará. Con todo ello ya es posible generar el terreno de manera correcta a partir de las señales de ruido, cada señal de ruido generará un terreno, pero este terreno será todo igual y repitiéndose periódicamente. Por tanto, para generar un terreno fractal que sea más interesante y detallado se necesitará de combinar las señales de ruido generadas. Un punto importante a decidir es cómo se combinarán estas señales. En el desarrollo de este trabajo se tomó la siguiente fórmula para combinar el ruido:

$$Ruido(x,z) = (Ruido1(x,z) + Ruido2(x,z) + Ruido3(x,z))/3 \qquad (5.2)$$

La cual es simplemente la media aritmética de los valores de los 3 ruidos. Más adelante en el desarrollo esta fórmula será cambiada por otra diferente que pro-

duzca resultados más realistas. En el momento actual de desarrollo, los terrenos se ven tal y como muestra la figura 5.6, siendo los tres primeros los generados por las señales de ruido y el cuarto la combinación de estos.

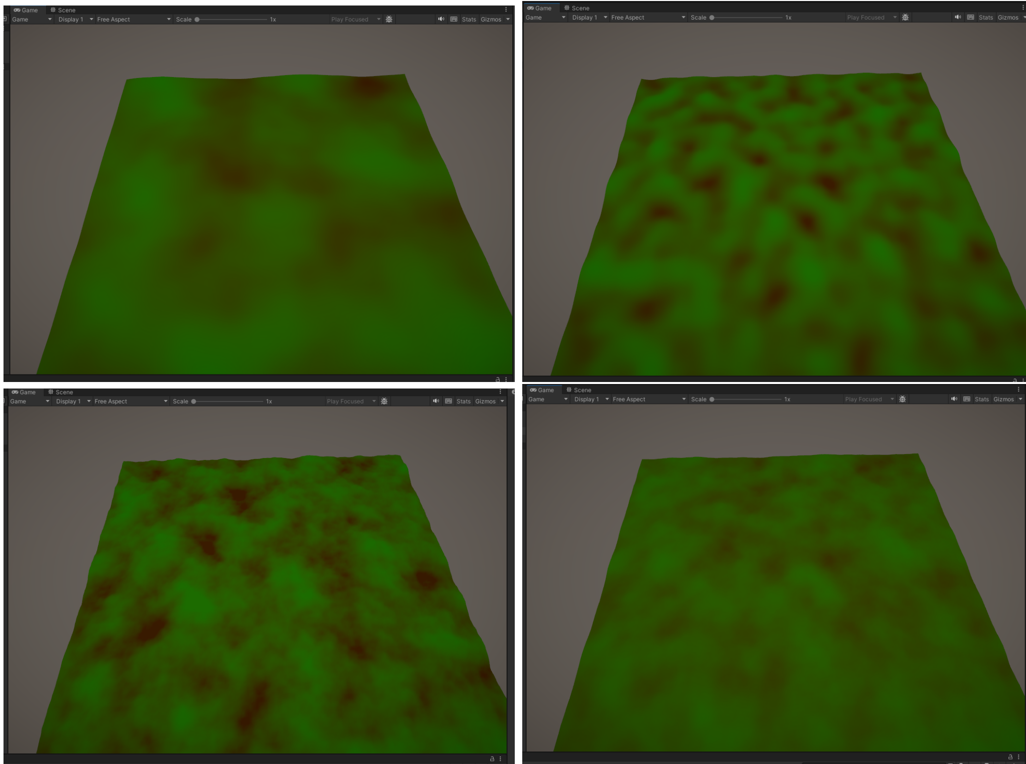

**Figura 5.6:** Terreno generado por las tres señales de ruido para una misma semilla y el terreno resultante de la combinación de estas señales

Para la siguiente versión del terreno se cambiaron diferentes acercamientos y parámetros. Primero se creará un enumerable para poder identificar los diferentes tipos de terreno de manera simple y explicativa. Por otro lado, se cambió la fórmula de la combinación para destacar diferentes aspectos de los ruidos en función de un nuevo factor. La nueva fórmula sería

$$Ruido(x,z) = a * (Ruido1(x,z) + Ruido2(x,z))/2 + (1-a) * Ruido3(x,z) \quad (5.3)$$

Entre otros de los cambios se cambió el color del shader a uno más cercano al amarillo para más adelante ser un color más similar al de las construcciones que se pondrán. Además, se simplificó y refactorizó el código para que este fuera más accesible y ordenado.

Otro cambio fue la introducción de las opciones, un objeto que deriva de ScriptableObject, una clase propia de unity, para poder ser fácilmente creado y así poder crear múltiples instancias para almacenar datos de diferentes terrenos. Estas opciones guardarían los datos sobre las dimensiones, la semilla y el material que utilizará la malla. Dados estos archivos de opciones, el generador deberá acceder al mismo para recoger todos los datos en sus propias variables.

Una última versión de la operación fue aplicada para poder destacar más los puntos donde los valores del ruido sean altos, creando así picos altos y planos

bajos, teniendo formaciones que parecen colinas más pequeñas alrededor, pareciendo una formación montañosa real. La fórmula para generarlo es la siguiente:

$$Ruido(x,z) = a * (0{,}5 * Ruido1(x,z) + 0{,}5 * Ruido2(x,z)) + (1-a) * Ruido3(x,z)^2 \tag{5.4}$$

y se puede apreciar en 5.7.

Para completar la última versión se configuró un conjunto de opciones en el inspector de unity para ser modificables en ejecución de manera simple. Estas opciones son las opciones del terreno deseado, la opción sobre que señal de ruido pretende utilizarse (o la combinación de las mismas), el máximo de la altura, el alfa de la operación y, finalmente, si se generará el terreno al ejecutar la aplicación o no. En caso de no hacerlo, este podrá ser generado accediendo a un método público que lo generará.

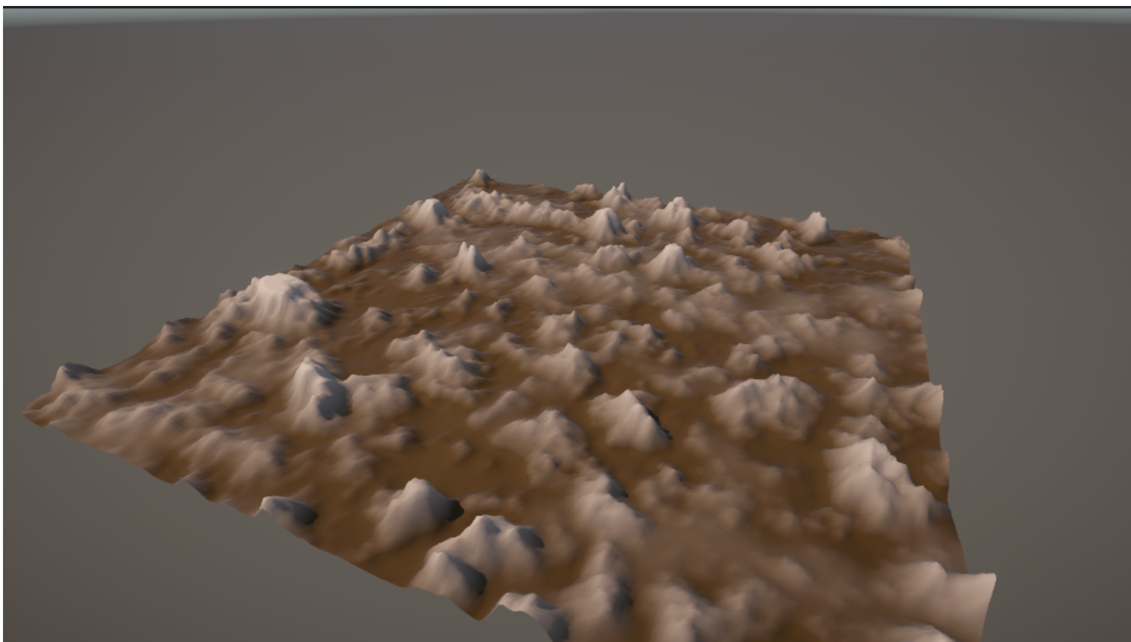

**Figura 5.7:** Terreno generado para la semilla 6494005 con un valor de alfa 0,25

## 5.3 Generador de construcciones

En el proceso de desarrollo del generador de construcciones se presentaron diferentes fases que finalmente culminaron en el objetivo deseado. Primeramente, una vez ya teniendo la gramática definida, tratándose esta de:

```
<Building> := <Wings>
<Wings> := <Wing> | <Wing> <Wings>
<Wing> := <Levels> <Roof>
<Levels> := <Level> | <Level> <Levels>
<Roof> := RoofA | RoofB
<Level> := Floor <Walls>
<Walls> := <Wall> | <Wall> <Walls>
<Wall> := Wall1 | Wall2 | WallW | WallWW | WallD | WallDW
```

se crearon todas las clases en el proyecto de Unity para representar a cada uno de los diferentes elementos que representaban, como se ve en 5.8. Cabe comentar que aquellos elementos que se encuentren en plural, como son "<Wings>" o "<Levels>", no tendrán una clase propia, sino que serán representados mediante listas en las clases de aquellos elementos que los contengan. Dicho esto, se crearon

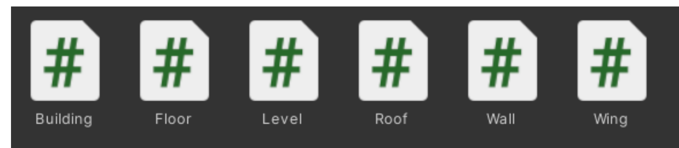


**Figura 5.8:** Listado de clases derivadas de la gramática

clases para construcción, ala, tejado, piso, muro y suelo. Las clases de construcción ala y piso contendrán 3 elementos fundamentales, siendo estos sus variables que los relacionen con el resto de elementos, un constructor y un método *toString* para poder comprobar su correcto funcionamiento. En cuanto a las variables cada clase contendrá las siguientes:

- Construcción contendrá un Vector2 para el tamaño que ocupará y una lista de alas
- Ala contendrá una lista de pisos, un tejado y un *RectInt* para delimitar el área que esta ocupa
- Piso contendrá el número del piso, una liste de paredes, un suelo y un *RectInt* para señalizar el área que ocupa

Todas estas variables tendrán otra variable pública asociada a ellas para evitar problemas de accesos y aportar una mejor encapsulación. A su vez, en el constructor de cada una de estas, se tomarán como argumentos los valores de todas las variables y estas se asignarán. Finalmente, el método *toString* simplemente imprimirá en formato de texto la estructura de la construcción Por otro lado, las clases de suelo, tejado y pared serán enumerables y estos deberán ser modificados dependiendo de los modelos que se estén usando, en este caso se trata de 6 tipos de paredes, 2 tipos de tejado y un tipo de suelo.

Una vez definidas todas las clases, se desarrollo una primera versión de la clase de generador de construcciones. Esta es una clase estática con todos los métodos necesarios para la correcta creación de la construcción. En concreto, contiene un método estático para generar cada uno de los elementos de la gramática, comenzando con la construcción. En esta primera versión, generar una construcción crea un objeto construcción y para la lista de alas que requiere llama al método de crear ala para generar una. A su vez, el método de crear ala toma los argumentos de espacio y genera un nivel y un tejado. El tejado generado en esta versión siempre es del primer tipo. Por otro lado, para generar el piso llama al método de generar suelo, que nuevamente genera un suelo del único tipo disponible, y también a generar paredes, que en esta versión solo genera del primer tipo. El resultado de este generador es siempre el mismo edificio, un edificio de un piso con cuatro paredes iguales, un techo y un suelo, tal como se puede ver cada uno de sus elementos en:

```
Building:((1.00,1.00);1)
   wing((x:0, y:0, width:1, height:1)):
   Level 1
     Walls: long1, long1, long1, long1
     Floor: floor1
   Roof: type1
```

Ya con la posibilidad de generar edificios simples el siguiente paso es construir el renderizador para poder colocarlos en la escena. El renderizador funcionará de manera similar al generador, basado en la gramática se le proveerá a un método Render de una construcción y este llamará a ′′renderizar ala" por cada ala disponible. Estas alas, a su vez, llamarán a ′′renderizar piso" y a ′′renderizar tejado". Cada uno de los pisos llamará a ′′renderizar piso" y a ′′renderizar pared". Existirá un método de ′′renderizar pared" específico para cada lado de la construcción, para colocarlo en su lugar adecuado y en orientación adecuada. Con el renderizador creado ya es posible colocar una construcción en el espacio, como es la construcción generada anteriormente renderizada en 5.9.

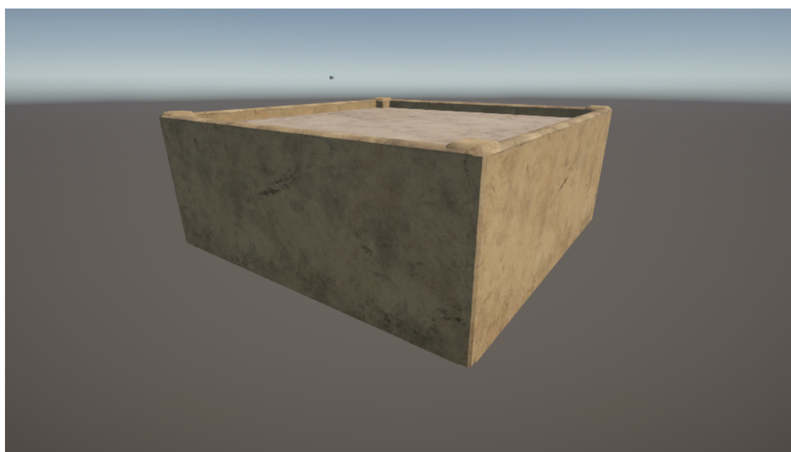

**Figura 5.9:** Construcción resultante del primer generador renderizada en la escena

Una vez funciona el building renderer, el siguiente paso es adaptarlo a poder renderizar edificios con más de una pared y más de un piso. Un aspecto importante a tener en cuenta es que para colocar las paredes correctamente en edificios

con más de una pared por lado se necesita conocer la distancia que ocupan dichas paredes. Esta distancia cambiará entre diferentes modelos y debe ser ajustada para no obtener resultados erróneos, como pasa en 5.10. En este caso se trata de una distancia de 0,42 unidades en los ejes X y Z y de 0.33 unidades en el eje Y, pero estos valores cambiarán para diferentes modelos. Para poder completar la generación de un edificio de mayor extensión se cambió en el generador el valor del tamaño que ocupa.

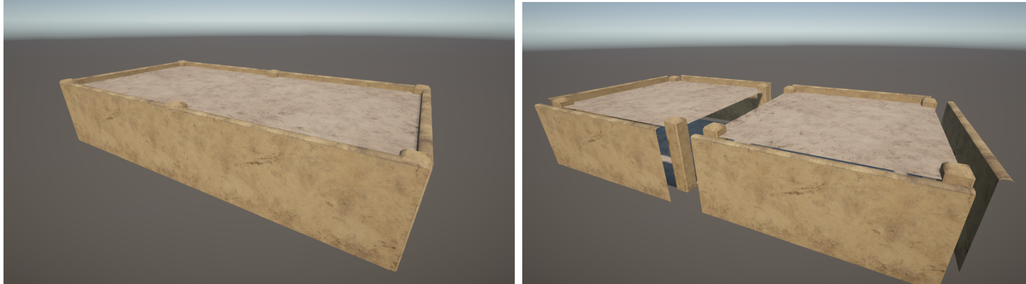

**Figura 5.10:** Construcción de dimensiones 2x1 y la misma construcción cuando los valores del renderizador no están correctamente ajustados

También es necesario ampliar la funcionalidad para poder generar edificios de más de un piso, no como está ahora. La solución a esto reside en por cada piso perteneciente al ala, renderizarlo indicando a el renderizador de las paredes en que piso se coloca, para así generarlo en la altura diferente que a este le procede. Dado que se conoce la altura de las propias paredes en el eje Y se necesitará multiplicar ese valor por el número del piso en el que se encuentre y colocar el tejado también en la altura máxima. Con esto se podrá generar edificios con cualquier número de pisos, y estos tendrán todos su suelo y sus paredes y el piso superior tendrá el tejado colocado encima, como en 5.11.

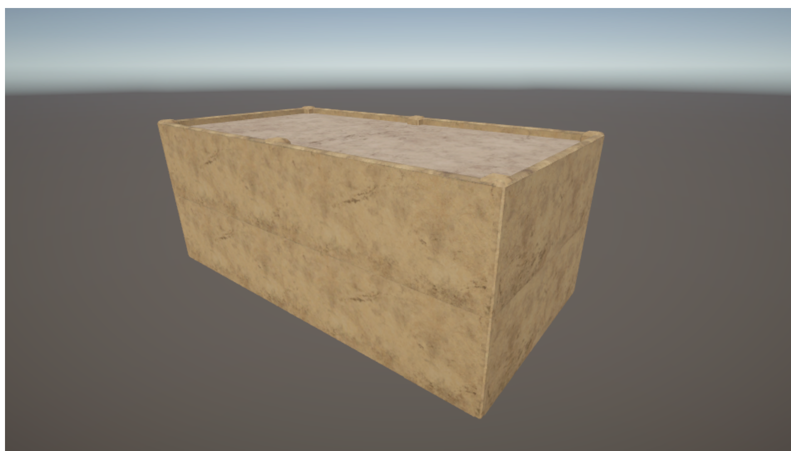

**Figura 5.11:** Construcción con 2 pisos de altura

El siguiente paso en el desarrollo del generador de construcciones es permitir en el renderizador renderizar diferentes alas. El objetivo es que cada ala sea totalmente independiente al resto, pero tenga un espacio delimitado dentro del espacio de la propia construcción. Esto también es necesario tenerlo en cuenta a la hora de trabajar con el generador, pero en el renderizador se deberá mover el centro de cada ala para representar el espacio que ocuparán. Finalmente, para

probarlo será necesario modicar el generador para que tenga más de un ala, como el edificio con 2 alas en 5.12.

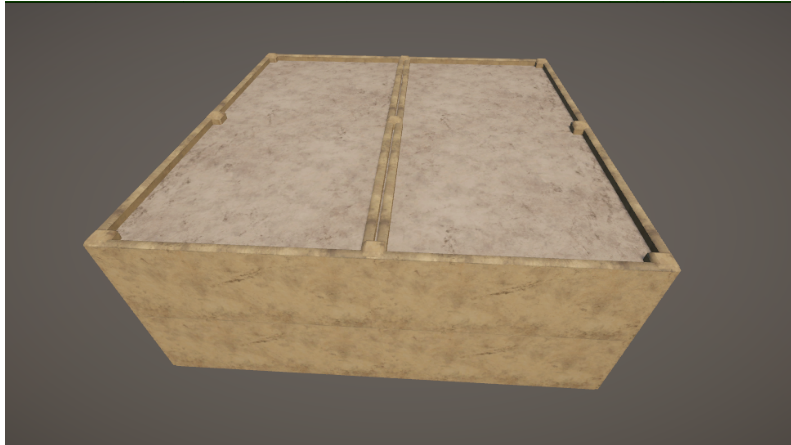

**Figura 5.12:** Construcción con 2 pisos de altura, dividida en 2 alas

Una vez expandido toda la funcionalidad en el renderizador, el siguiente objetivo es añadir un objeto de opciones que contenga todas las variables necesarias a dar al generador. Estas opciones, al igual que para el terreno, serán una clase que heredará de la clase ScriptableObject de unity, así pudiendo crear y guardar instancias del mismo fácilmente. En esta clase se almacenará lo siguiente: el tamaño del edificio, el número de alas y el número de pisos. Con esto, se debe refactorizar el generador para acomodar sacar la información directamente de las opciones. El resultado es visible en 5.13. Con esto se podrán crear nuevos objetos de opciones con valores diferentes para cambiarlos a placer de manera sencilla.

Una vez las opciones ya son implementadas, se entrará en la tarea de crear diferentes estrategias para generar los diferentes tipos de elementos de la gramática. La razón de ser de las estrategias es para añadir una funcionalidad modular al generador de construcciones, donde puedan convivir diferentes estrategias para los diferentes elementos de manera independiente. Las estrategias cubrirán únicamente la generación del elemento específico al que se refieran, dado que no se pretende que superpongan la funcionalidad de otras. Se crearán estrategias para los siguientes elementos: las alas, los pisos, los tejados y las paredes. No es necesario crear en este caso ninguna estrategia para los suelos, aunque sí que podría hacerse, por ejemplo, una estrategia que rotara el suelo de manera aleatoria. Para la construcción no es necesaria crear una estrategia dado que las propias opciones actúan como una estrategia, definiendo cómo se tienen que generar los edificios.

Centrándose en las estrategias propias de los diferentes elementos, todas seguirán la misma estructura. Existirá una clase abstracta de estrategia para cada uno (ya sea pared, tejado, etc), la cual contendrá un método abstracto para generar estos elementos. Heredando de estos, existirán todas las estrategias disponibles para estos elementos. Como mínimo, para cada estrategia existirá una predeterminada. Todas estas estrategias serán almacenadas para cada construcción a generar en las opciones, y se entenderá que si alguna de las variables de estas no está asignada esta será la opción por defecto. Con ello, el funcionamiento será el siguiente. Cuando se llame al método generar construcción del generador, este llamará al método generar alas de la estrategia asignada, que a su vez llamará a

los métodos de generar tejado y generar pisos de sus respectivas estrategias en las opciones y así sucesivamente. En el caso anterior de no existir una estrategia asignada, se creará una instancia de la misma en el momento de necesitarla.

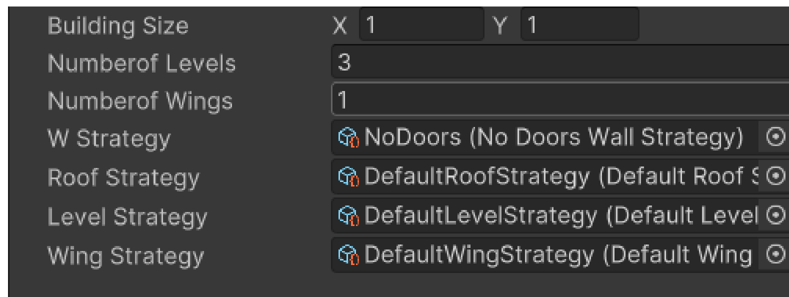


**Figura 5.13:** Interfaz de las opciones de una construcción

Entrando en detalle en el funcionamiento de cada una de las estrategias, comenzando con la estrategia de las paredes, encontraremos la estrategia predeterminada y una estrategia adicional. La estrategia predeterminada instancia toda pared como la primera pared disponible, siendo entonces todas las paredes generadas planas e iguales. Por otro lado, la estrategia adicional sigue un acercamiento basado en el nivel en el que se encuentre la pared. Si la pared se encuentra en el nivel inferior, generará una pared aleatoria entre todas las disponibles. Si esto no es así, generará una pared aleatoria excluyendo aquellas que contengan una puerta, puesto que sería imposible entrar por ella si esta construcción fuera real, como se puede apreciar en 5.14. El correcto funcionamiento de esta es dependiente de la posición de las paredes en el listado de referencias del generador, dado que funciona generando solo las paredes en los índices de 0 a 3, cuando las paredes con puerta están en las posiciones 4 y 5.

**Figura 5.14:** Construcción con 3 pisos de altura y la estrategia de las paredes aplicada

Para las estrategias de las alas, se deberán implementar tanto un método para generar un ala como para generar múltiples alas, respondiendo así a la definición en la gramática, como demuestra la figura 5.15. La estrategia por defecto divide

el espacio en partes de igual tamaño, incluso desechando espacio si este sobrara. Para ello toma la dimensión en X y la reparte entre todas las alas, dejando la dimensión en Y intacta. Tras ello, cada una de estas alas llaman a generar el ala, que llama al método generar pisos del propio generador. Por otro lado, existe otra estrategia para las alas, la cual define la longitud de las mismas en el eje Y de manera aleatoria entre 1 y el valor estipulado. De esta manera si insertamos un valor de 5 en la Y, puede ser que esta ala sea de longitud 1 a 5 en este eje. Además, al generar los pisos esta estrategia también genera una cantidad aleatoria de pisos entre 1 y el valor estipulado.

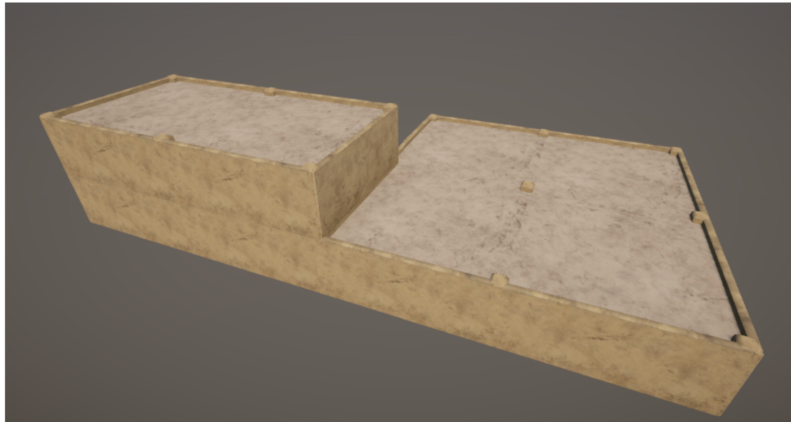

**Figura 5.15:** Construcción con 2 alas de diferentes dimensiones, una con 2 pisos y otra con 1

En relación a las estrategias de los tejados, todas deberán implementar un único método de generar tejado. La versión por defecto siempre generará el mismo tipo de tejado. La otra estrategia disponible expande sobre esta funcionalidad, generando uno entre dos tejados y rotándolos de manera aleatoria, como en la construcción en 5.16. Una funcionalidad no presente en este acercamiento es la capacidad de aplicar diferentes estrategias a los tejados de las diferentes alas, por lo que tendrán todas el mismo funcionamiento de generación de tejados.

Finalmente, para la estrategia de los niveles actualmente solo existe una, la estrategia por defecto. La estrategia por defecto únicamente genera las paredes de manera normal, llamando a la estrategia de paredes para completar esta función. Uniendo todos los cambios del generador de construcciones con la funcionalidad de las estrategias permite combinar y expandir estas mismas para conseguir el funcionamiento deseado. Dado que cada una de estas estrategias funciona de manera independiente y llama a la siguiente si esta existiera, no existe ningún riesgo en combinarlas para crear edificios más interesantes, como en la figura 5.17.

**Figura 5.16:** Construcción con 4 tejados generados mediante la estrategia adicional

**Figura 5.17:** Construcción generada mediante la combinación de las 3 estrategias adicionales explicadas en esta sección

## 5.4 Generador combinado

Centrándose en el desarrollo del generador combinado, este desarrollado en diferentes pasos para conseguir la funcionalidad deseada. Primeramente, como este debe acceder a las funcionalidades de ambos generadores, de construcciones y de terreno, se decidió que la clase que lo implementara heredaría de la clase del generador del terreno y accedería al generador de construcciones mediante una referencia estática. Una vez esta cuestión solventada, es necesario ampliar todas las funcionalidades del generador del terreno para poder generar las construcciones.

El primer paso para generar las construcciones sobre el terreno es declarar una nueva señal de ruido para definir los puntos donde se deberán incluir estas. Inicialmente esta sería como la señal de ruido 2 del generador del terreno, ligeramente diferente. Más adelante en el desarrollo se decidió cambiarla por otra señal de ruido con parámetros diferentes. Los resultados generados por el primer terreno, como se aprecia en 5.18, son muchas construcciones diferentes esparcidas en pequeños grupos a lo largo de todo el terreno. Por otro lado, los resultados generados por la segunda señal de ruido son un gran conjunto de edificios unidos, dando apariencia de asentamientos a lo largo del terreno como en 5.19. Por tanto las señales de ruido se quedarán tal que:

| | Frecuencia | Lagunaridad | Número de octavas | Persistencia |
|---|---|---|---|---|
| Ruido inicial | 0.065 | 2 | 3 | 0.1 |
| Ruido final | 0.045 | 2 | 3 | 0.3 |

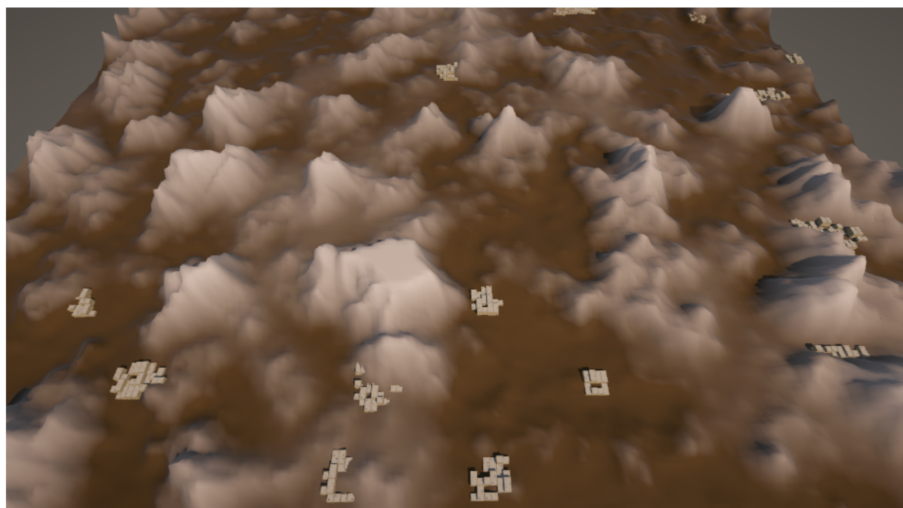

**Figura 5.18:** Terreno con construcciones generadas mediante la primera señal de ruido.

Una vez definida la señal de ruido se deberá crear un método como el que tiene la clase de generador de terreno para poder generar el terreno con construcciones incluidas. Aunque primero se trató de desarrollar un método sobrecargado del propio del generador del terreno, finalmente se decidió por un método independiente del generador combinado. Este método seguirá el mismo proceso del generador del terreno, esto es iterar por todos los puntos definidos, creando los índices y los vértices, y añadirá diferentes funcionalidades para poder colocar

edificios sobre el mismo. El primer paso para esto será comprobar si el punto en la señal de ruido para el que queremos crear el vértice supera un mínimo establecido en una variable. Si esta condición se cumple, se podrá pasar a la generación de construcciones.

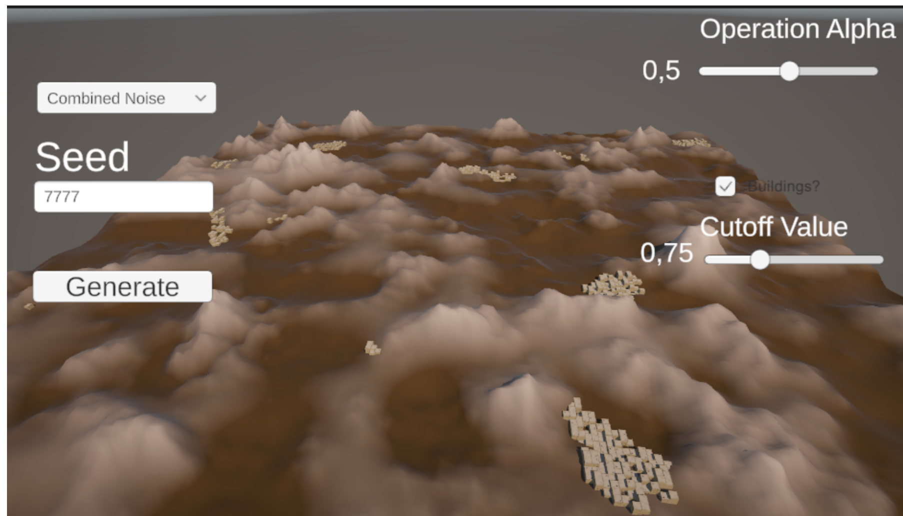


**Figura 5.19:** Terreno con construcciones generadas mediante la segunda señal de ruido.

Las construcciones generadas por el generador tienen la particularidad de que, debido al paquete de modelos presente, no ocupan un espacio de 1x1 unidades, sino de 0.82x0.82. Ya que se desea que estas construcciones se encuentren alineadas en el espacio, se escalará la construcción por un factor igual a 1/0.82, aunque podría dejarse este tamaño más pequeño para lograr un efecto diferente, pareciendo el espacio entre las construcciones calles tal y como se muestra en 5.20.

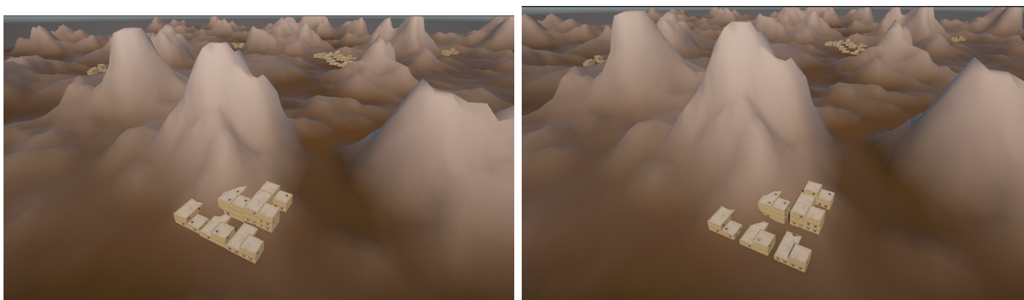

**Figura 5.20:** Conjunto de construcciones con y sin escalado

La siguiente cuestión a abordar se centra en el posicionamiento de las propias construcciones. Si se genera el terreno con construcciones, colocando estas en sus posiciones (x,y,z) basándose en el valor de la posición del vértice colocado en ese punto se podrá observar que algunas construcciones se generarán levantadas del suelo, al tomar como punto de referencia el punto máximo del suelo donde se construiría. Esto lleva a tener que decidir una estrategia para generar las construcciones que minimicen este problema. Existen 2 acercamientos:

- Hacer una media de altura de todos los puntos en que se sitúa la construcción y hacer de la media la altura.

- Hacer una búsqueda del valor mínimo de todos los puntos y tomar este como altura.

Ambos de estos acercamientos tienen sus beneficios y desventajas. Calcular la media hará que todos los puntos estén lo más cerca posible de su altura real, pero pueden seguir pareciendo edificios flotantes y cambios repentinos de altura pueden sesgar esta media negativamente. A su vez, calcular el valor mínimo hará que ningún punto se sitúe sobre su altura real, pero puede generar edificios que se encuentren enterrados bajo el terreno. Finalmente se tomó el segundo acercamiento para evitar la aparición de edificios flotantes sobre el terreno, alegando un aspecto más realista a pesar de que algunos edificios se encuentren soterrados. El resultado es visible en el terreno, como en 5.21.

**Figura 5.21:** Resultado de ejecutar el generador combinado tomando la altura de los edificios como la mínima altura que ocupan

Otro punto a tener en cuenta cuando generando las construcciones es tratar de no generar estas sobre otras ya existentes. Dado el funcionamiento del código únicamente se requiere del punto inicial para generar una construcción, sin tener en cuenta la posibilidad de ya existir una en una posición que ocuparía. Debido a esto se creará una lista para apuntar todas las posiciones reales donde una construcción se halle para así no generar nada en esos puntos. De la misma manera, si se quiere volver a generar un terreno se deberá borrar de la escena toda construcción habida para dar cabida a las nuevas generadas. Para ello se creará una lista con todas las estructuras que se generen y se destruirán todos los elementos de esta lista cada vez que se desee generar un nuevo terreno con construcciones. Si no se restringe, pueden darse resultados como en la figura 5.22, donde la ventana en el edificio de la derecha está superpuesta.

Finalmente se debe explicar el hecho de la existente limitación de altura a la hora de generar construcciones. Dado la fórmula final para la generación del terreno combinado, el terreno resultante es uno con planos y bajos, por un lado, y picos y montañas muy empinadas. Si se tratase de generar construcciones en estas alturas, no importa que intento o algoritmo se utilizara, el edificio aparecería flotante, como en 5.23. Dado también que al ser cuadrático el factor de el tercer

**Figura 5.22:** Resultado de ejecutar el generador combinado sin restringir la repetición de construcciones en un punto

ruido del terreno, existirán menor cantidad de valores cercanos al punto intermedio que con un acercamiento lineal, dejando la mayor parte del terreno en valores más bajos. Así pues, al limitar la altura de la generación lo que realmente se consigue es que las construcciones se generen en estos niveles más bajos, donde la posibilidad de aparecer edificios flotantes es prácticamente nula.

**Figura 5.23:** Resultado de ejecutar el generador combinado sin restringir la altura de generación de construcciones.

Teniendo todos estos puntos en cuenta será posible la correcta generación del terreno con construcciones sobre las mismas, completando así el desarrollo de este generador combinado. El resultado de una ejecución de tal es visible en la figura 5.24.

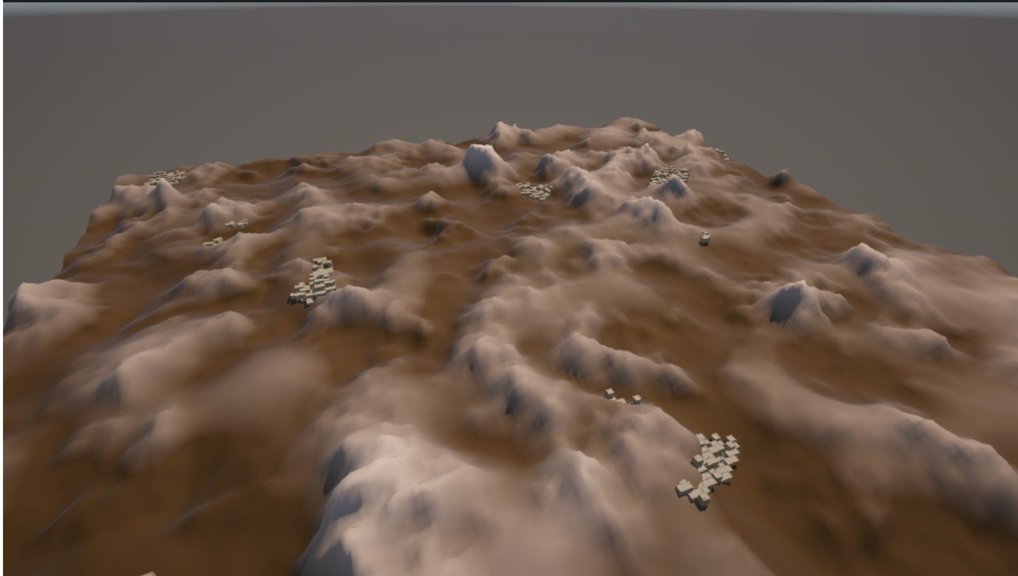

**Figura 5.24:** Resultado de ejecutar el generador combinado.

## 5.5 Demo completa

Una vez ya completado el desarrollo del generador combinado es preciso crear una demo o herramienta para poder mostrar las capacidades de la misma. Esta demo permitirá acceder y modificar todos aquellos parámetros relevantes a la hora de generar tanto el terreno, como las construcciones y la combinación de ambos.

Comenzando con el generador de terreno, deberemos tener en cuenta los siguientes valores modificables:

- El tipo de terreno.
- Las dimensiones del terreno.
- El material del terreno.
- El valor del alfa para la combinación.
- La semilla del terreno.

Los 3 primeros valores están contenidos en las opciones del terreno que se deben suministrar al generador, mientras que el valor de alfa no está incluido. Dada la naturaleza de esta demo, existirá en esta un único objeto de opciones de terreno y solo será posible cambiarlo desde el editor, no desde la aplicación. Sin embargo, dado que el tipo del terreno a generar es interesante para cambiarlo desde la ejecución, se incluirá como valor modificable en la demo. Esto deja como opciones modificables para el terreno: el tipo, el alfa y la semilla. Se pueden identificar los elementos en la figura 5.25.

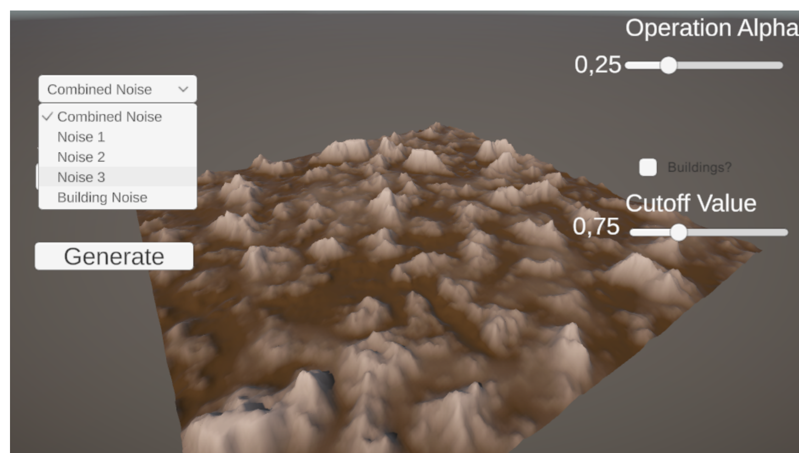


**Figura 5.25:** Captura de la demo detallando los diferentes tipos de ruidos seleccionables.

Por otro lado, el generador de construcciones contiene diversos valores modificables, siendo estos la semilla nuevamente y todo aquello contenido en las opciones de construcción, dimensiones, número de niveles, estrategias y número de alas. Así mismo, se probará y incluirá un objeto de opciones que no será modificable desde la demo. Un resultado de las mismas es visible en 5.26. Estas opciones serán antes probadas y elegidas las más apropiadas. De esta manera, el único valor modificable desde la demo de este generador será la semilla.

**Figura 5.26:** Captura de la demo detallando los tipos de construcciones generados.

En cuanto al generador combinado, este tendrá varios valores modificables desde la propia interfaz de la demo. En primer lugar, existirá una opción de activar o desactivar la generación de construcciones, lo cual sería el equivalente a decidir si utilizar el generador de terreno o el combinado como en 5.27. Si se decide utilizar construcciones, estas solo se generarán sobre el terreno combinación, no sobre ninguno otro singular. Además, existirá un valor modificable para decidir cuantas construcciones se deben generar, si el valor del ruido de estas supera un umbral. Dado que ambos generadores que constituyen este requieren de una semilla, este tendrá un valor modificable de semilla, el cual será el mismo para ambos generadores subyacentes.

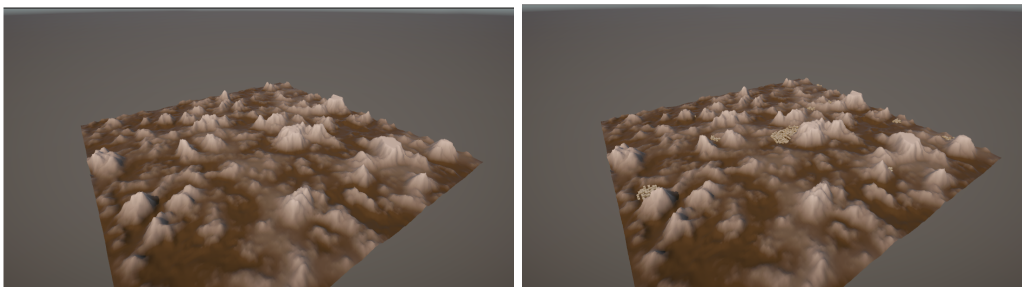

**Figura 5.27:** Un mismo terreno con y sin construcciones.

Una vez decididos todos los valores a tener en cuenta a la hora de interactuar con la demo. Unity incluye una serie de componentes preparados para funcionar de manera interactiva, como botones o barras. Tras ver las necesidades del generador estos son los elementos de la demo.

- Un *Dropdown* para elegir el tipo de terreno.
- Un campo de entrada para elegir la semilla.
- Una barra deslizante para seleccionar el alfa, en intervalos de 0.01.
- Una barra deslizante para seleccionar el valor umbral, en intervalos de 0.01.
- Una casilla a marcar para generar construcciones o no.

- Un botón para generar el terreno dado los valores previstos.

Todos estos son visibles en 5.28.

**Figura 5.28:** Captura de la demo con los diferentes elementos interactuables.

Una vez se presione el botón de generar, la demo primero comprobará si se desean construcciones o no. Tras esto, verá la opción del terreno seleccionada y se lo mandará a generar al generador combinado. Por otro lado, los parámetros de alfa, semilla y umbral se pueden cambiar previamente a la generación, pero una vez se presione el botón se tomarán los valores actuales para la generación. El resultado final es apreciable en 5.29.

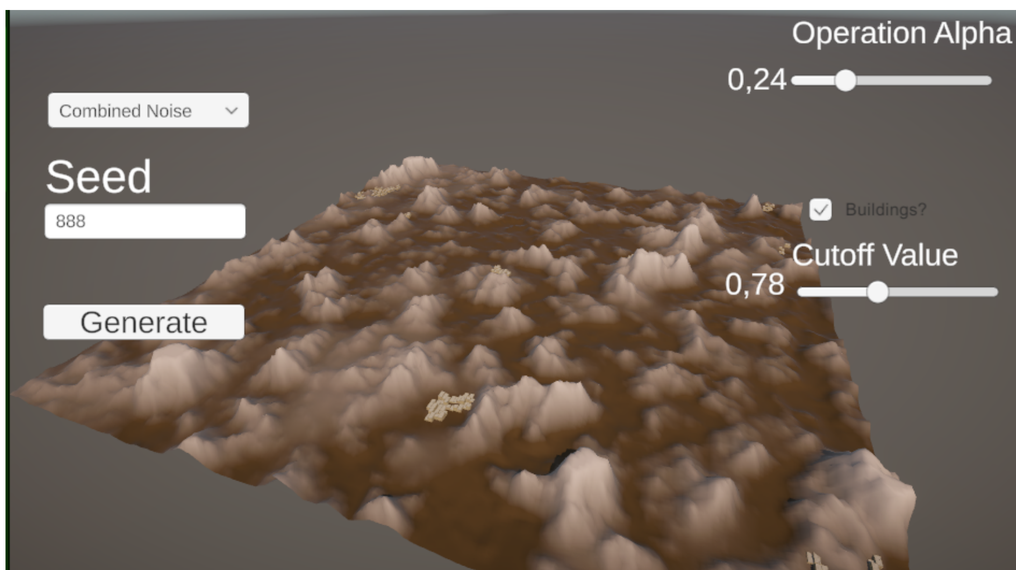


**Figura 5.29:** Captura de la demo en su estado final.

# CAPÍTULO 6
# Pruebas

Ya desarrollada la aplicación, el siguiente paso a realizar es el de verificar el correcto funcionamiento de la misma y decidir sobre que parámetros debería actuar la misma.

Es preciso comprobar que todas las posibles configuraciones de valores funcionen, demostrando que todo el trabajo realizado por debajo de esta demo funciona correctamente. Primero se realizarán una serie de pruebas relacionadas con la corrección del resultado y, a continuación, se aplicarán otras pruebas basadas en la calidad del resultado generado.

## 6.1 Corrección

Uno de los casos más importantes a evitar es el incorrecto funcionamiento de las herramientas, ya sea devolviendo errores o generando resultados inesperados. Para ello deberán ser probadas diferentes configuraciones para concretar qué funciona y qué necesita ser acotado a unos valores determinados.

Comenzando por el tamaño del terreno, ya en la sección anterior se detectó un claro problema: el motor de Unity no soporta generar un terreno tan grande como 300x300 en una iteración del mismo. El resultado será un terreno con vértices no generados, como en 6.1, y, por lo tanto, los triángulos estarán desalineados al no corresponder los vértices. Este problema es fácilmente evitable, es necesario limitar las dimensiones del terreno a igual o menor al valor más grande probado, siendo este 62500 o 250x250. En este mismo campo, el generador del terreno no funcionará para dimensiones menores o iguales a 0 ni para dimensión de 1 en cualquiera de los ejes. Esto se debe a que el algoritmo que define los índices se saltará la última fila de vértices definidos y, por lo tanto, al solo existir una fila nada será definido. En conclusión, el generador de terreno no puede generar una línea ni un terreno demasiado grande.

El siguiente punto a tener en cuenta es la importancia de aplicar un material al terreno en la generación. En las opciones del terreno el material es un campo importante de llenar dado que si se trata de ejecutar el generador de terreno sin el mismo definido el terreno resultará sin textura, siendo imposible de distinguir partes del mismo.

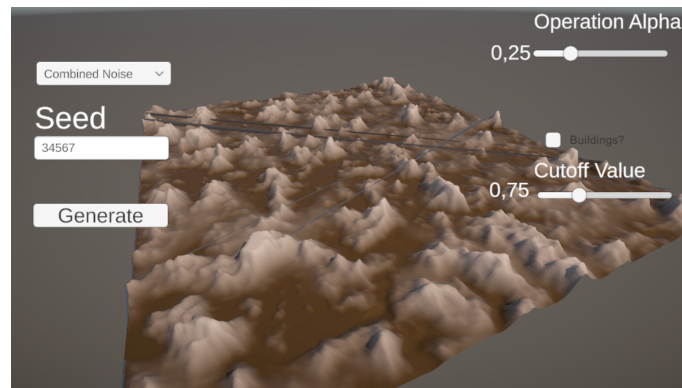


**Figura 6.1:** Errores visibles en la generación al tratar de generar un terreno de 275x275

Otra cuestión a probar es la correcta generación desde la demo de todos los tipos de terrenos y que estos sean reconocibles en el terreno de la combinación. La propia demo tiene una selección de opciones para generar los diferentes tipos de terreno, así pues, seleccionando estas opciones y generando el terreno debería generar los terrenos que definen las señales previamente definidas. Tras comprobar esta cuestión desde la demo desarrollada el resultado es satisfactorio apreciable en 6.2, todos los terrenos se generan de manera correcta.

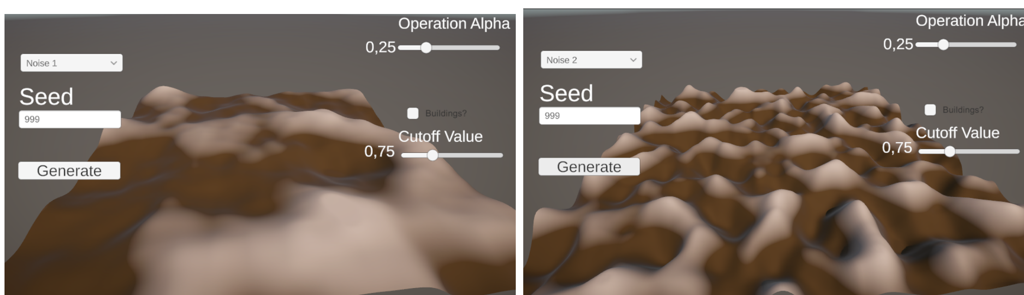


**Figura 6.2:** Señales de ruido siendo generadas correctamente en la demo.

Ahora con las construcciones, se debe comprobar que todas las combinaciones posibles de valores funcionan correctamente, y que estos son representados en el resultado final.

Primeramente, serán comprobados los valores de dimensiones pisos y alas, que estos funcionen tal y como fueron definidos en apartados anteriores. Para comprobaremos podremos cambiar estos valores a otros más extremos, para encontrar los problemas que puedan surgir. Tras realizar una serie de pruebas con estos valores se puede comprobar que estos funcionan independientemente de los valores suministrados, si bien la ejecución sufre de más necesidad de recursos al haber una gran cantidad de triángulos siendo cargados por la pantalla, como ocurre al cargar la construcción de la figura 6.3.

Sobre las estrategias extra debemos ver individualmente que:

- Se generen paredes aleatorias en el edificio y no se generen puertas en los pisos superiores
- Se generen alas de diferentes tamaños y alturas

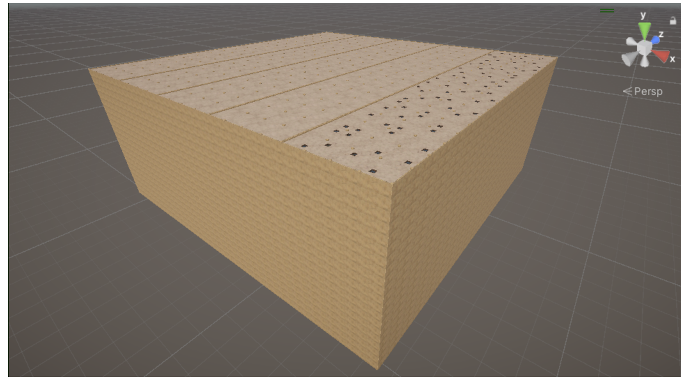


**Figura 6.3:** Resultado de generar un edificio de 20 pisos de altura y dimensiones 20x20, siendo dividido en 5 alas

- Se generen tejados diferentes en las diferentes alas

Y una vez estas sean correctamente comprobadas todas deberán estar presentes en una misma construcción. Es importante comprobar el correcto funcionamiento de estas antes de colocar las construcciones en el terreno, pues un problema a esta pequeña escala podría reproducirse muchas veces en el terreno completo. El satisfactorio resultado es visible en 6.4.

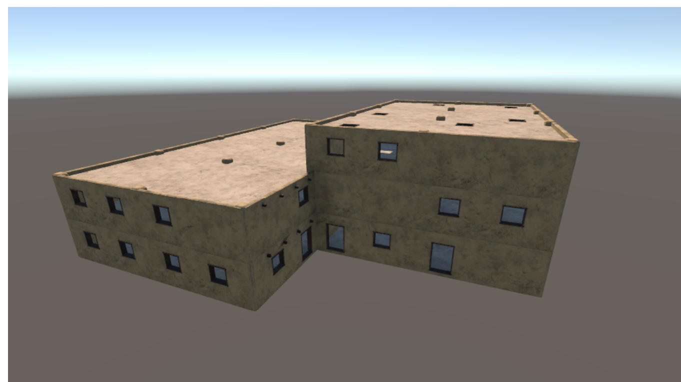

**Figura 6.4:** Resultado de generar un edificio con todas las estrategias extras siendo aplicadas al mismo tiempo

Una vez comprobado que tanto el generador de terreno como el de construcciones funcionan correctamente es necesario probar el generador combinado mediante la demo. Dos puntos importantes a tener en cuenta son la cantidad de construcciones a generar y la correcta generación de las mismas.

Primero debemos comprobar que las construcciones solo se generan cuando se selecciona la opción de ruido combinado y no se genera sobre las señales de ruido básicas. Esto se cumple así y es fácilmente replicable en la demo. Sin embargo, un caso especial ocurre cuando se pone un número muy alto en el campo de semilla. Se hizo una prueba con el número

`3212345678900987651234567890098765432123456789098765423456789`

, siendo este un número de 60 dígitos. El resultado era las mismas señales de ruidos que para otros valores de similar tamaño, al haber *overflow* en la variable entera, incapaz de almacenar este número sin perder datos.

El otro punto a tener en cuenta es el valor del umbral de generación de construcciones. Demasiadas construcciones en pantalla podrían ralentizar la ejecución de la demo o pararla definitivamente, así que se debe tener en cuenta este dato. En una ejecución regular, al generar el terreno combinado se crean 318800 triángulos (Unity crea más vértices a partir de los definidos en el generador para aumentar la resolución del terreno) sin haber generado ninguna construcción. Si generamos construcciones con un umbral de 0.9 este valor pasa a 726000, duplicando la cantidad de los mismos. Aun con esta cantidad de triángulos la velocidad del editor se mantiene constante, alrededor de 200 FPS. En una siguiente prueba se cambia el valor del umbral a 0.75, habiendo 2,2 millones de triángulos. Aun habiendo alrededor de 3 veces más triángulos generados, la velocidad del editor oscila entre los 60 y 100 FPS, siendo estos una velocidad adecuada para la demo. Finalmente, se prueba con umbral de 0.64. La cantidad de triángulos llega a 4.5 millones y la velocidad del editor baja a alrededor de 40 FPS, siendo las ralentizaciones visibles, aunque no grandes. Dado que la cantidad de triángulos en memoria y la velocidad del propio editor están bajando drásticamente con cada cambio del umbral, no debería este ser menor al valor de la última prueba realizada para mantener una velocidad adecuada y la memoria con espacio. Un ejemplo de estas mediciones de datos es la figura 6.5, donde en la esquina inferior derecha se presentan estos datos.

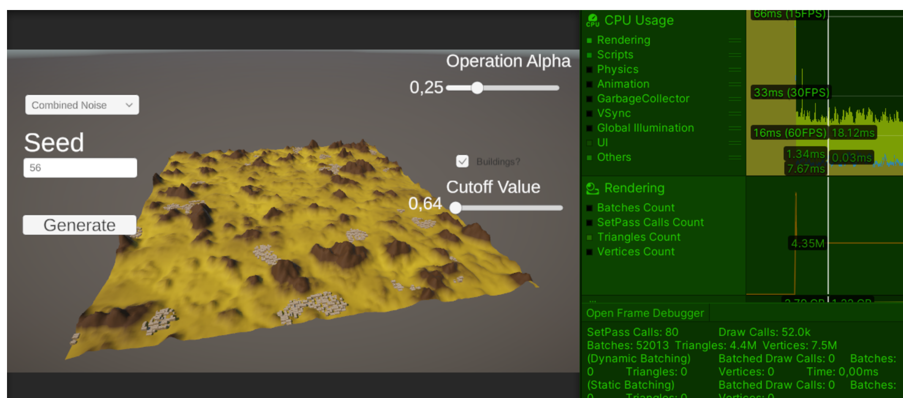


**Figura 6.5:** Medición de la velocidad del editor de Unity utilizando la herramienta Profiler

## 6.2 Calidad

Una cuestión relevante a la hora de generar contenido procedural es la calidad del mismo. No tiene ningún interés generar contenido si todo aquello generado es blando y indistinguible del resto. Es por ello que en esta sección se pretende cambiar diferentes parámetros para conseguir un resultado más detallado e interesante.

Un primer punto a definir es como generar las construcciones para que sean interesantes. Las construcciones con las opciones predeterminadas son todas edificios cuadrados del mismo tamaño, sin ninguna clase de detalle. Es por ello que las estrategias adicionales deberían ser empleadas, otorgando más detalle al resultado generado. Otro punto a tener en cuenta sobre estas es las dimensiones de

las mismas al colocarlas en el terreno. Una construcción muy grande ocupará el terreno de otras y así sucesivamente. A la hora de generar construcciones en el terreno es mejor crear construcciones más pequeñas, dejando que así se generen unas al lado de otras otorgando un aspecto más natural de villa. Es por ello que las opciones finales de la demo respecto a construcciones serán construcciones de 2x2 con 3 pisos de altura máxima, divididos en 2 alas, aplicando las estrategias extras para tejados, alas y paredes, dándo el resultado de 6.6. Por parte del te-

**Figura 6.6:** Resultado de generar construcciones sobre un terreno con la configuración mencionada

rreno, es posible generar diferentes configuraciones en función del valor del alfa de la combinación. Con un valor de alfa igual a 0, el terreno resultante será el cuadrado de la tercera señal de ruido, mientras que con un valor de 1 este será la media entre las 2 primeras señales de ruido. En esta prueba se pretende encontrar algún valor entre estos que dé lugar a una configuración de interés. Se han probado diferentes valores y de ellos se han sacado conclusiones:

- Con valores menores de 0.15 para alfa el terreno no toma apenas valor de la combinación de los 2 primeros, generando un terreno como la tercera señal cuadrada ligeramente suavizada.
- Con valores mayores de 0.6 y menores de 0.8 se pueden observar las diferentes facultades del terreno dadas las señales, pero el resultado es demasiado suavizado.
- Con valores mayores de 0.8 el terreno es prácticamente solo la combinación de las primeras dos señales.
- Los valores entre 0.15 y 0.6, pero especialmente entre 0.3 y 0.5 resaltan en gran medida las facciones del terreno, se pueden ver picos generados por la tercera señal y colinas alrededor generadas por las otras señales.

Los resultados de las pruebas que muestran estos son visibles en las figuras 6.7 y 6.8. Teniendo en cuenta estos resultados de las pruebas podemos concretar una serie de valores que generan un terreno y construcciones más interesantes y detalladas, produciendo un resultado realista para poder ser utilizado en otros proyectos diferentes.

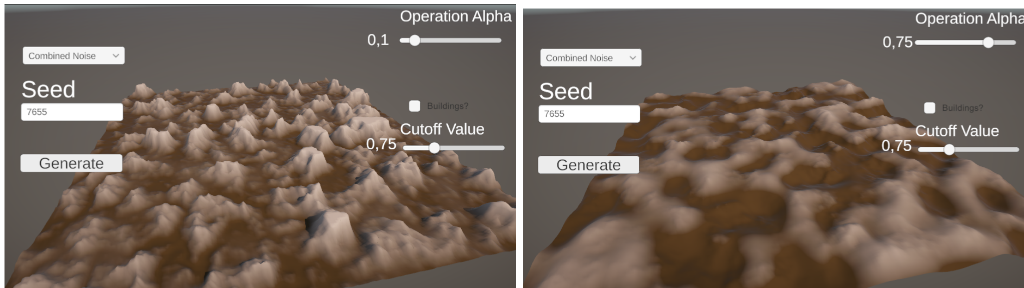


**Figura 6.7:** Terreno generado para valores de alfa 0.1 y 0.75 respectivamente

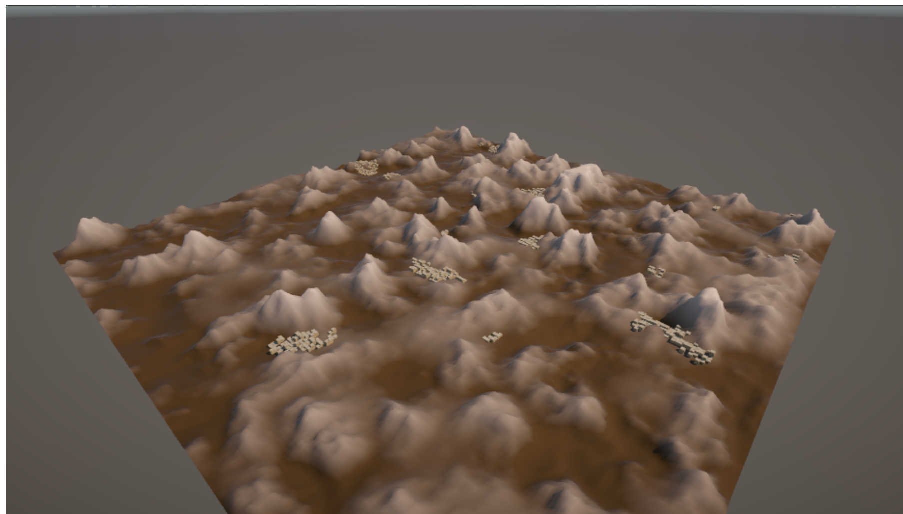

**Figura 6.8:** Terreno generado para valores de alfa 0.34 y construcciones con umbral de 0.75

# CAPÍTULO 7
# Conclusiones

En conclusión, en este trabajo se han planteado y alcanzado los siguientes objetivos:

- Primeramente, se ha investigado sobre diferentes técnicas de generación procedural y se han seleccionado varias de estas para crear diferentes generadores, siendo estas el uso de señales de ruido y el uso de gramáticas.
- Para cada una de estas técnicas, se ha creado un generador procedural que las aplique, siendo estos el generador de terreno y el generador de construcciones.
- Se han combinado ambos generadores para conseguir una funcionalidad conjunta, generando las construcciones sobre el terreno.
- Finalmente, se ha desarrollado una demo que permite visualizar y probar los resultados de los generadores creados.

Es posible concretar que se han cumplido los objetivos planteados en la primera sección del trabajo de manera satisfactoria. Para completar estos ha sido necesario conocer sobre las señales de ruido y gramáticas, utilizar librerías como LibNoise para generarlas fácilmente y organizar el desarrollo en función de la gramática para el generador de construcciones. Si bien ya se conocía sobre las gramáticas previamente al comienzo del trabajo, el autor no las había aplicado de esta manera hasta el momento.

El desarrollo de todos los generadores también ha conllevado una cantidad de problemas debido a limitaciones y especificaciones del propio motor utilizado, como son las limitaciones en la generación del terreno o la distribución de las paredes en las construcciones. La solución para estas fue, por un lado, limitar las dimensiones del terreno y alinear las paredes tomando las medidas específicas de las mismas. Otros problemas menores surgieron con los materiales, ya sea que no estaban cargados o que no reflejaban las sombras creando un aspecto inadecuado, y estos fueron solucionados descargando un material que aceptara las luces y sombras.

A su vez, durante el desarrollo de este proyecto se han aprendido diferentes cuestiones centradas en diferentes dimensiones. Por un lado, de organización y metodología, con conceptos como ser organizado y consistente, preparar el trabajo antes de comenzarlo y planear en el tiempo. Por otro lado, de conceptos se

aprendió mucha información sobre el tema central del TFG, la generación procedural, así como de señales de ruido, generación de gráficos en 3D y las posibilidades del motor de Unity. También se profundizó más en el uso de gramáticas, un tema del cual ya se tenía un conocimiento superficial. Finalmente, también hubo gran parte de aprendizaje referente a la integración y uso de herramientas externas,como son las librerías.

## 7.1 Relación del trabajo con estudios cursados

En cuanto a cuáles disciplinas han sido aplicadas en el desarrollo de este TFG, es posible nombrar diferentes de ellas para las diferentes partes.

Primeramente, para el desarrollo del generador de terreno es claramente apreciable la aplicación de los conocimientos adquiridos en asignaturas relacionadas con la programación gráfica, hablando de vértices, triángulos, generar mallas en 3D, shaders y materiales. Todos estos puntos nombrados son de gran relevancia en el desarrollo del mismo y han sido nombrado múltiples veces en la propia memoria.

Por otro lado, diversos conceptos centrados en programación algorítmica fueron aplicados en el diseño y desarrollo de los diferentes generadores, incluyendo el uso de diversas estructuras de datos para trabajar con los diferentes objetos y componentes desde el código de manera útil y sencilla.

De la misma manera, múltiples diferentes conceptos de las asignaturas de desarrollo y diseño de videojuegos fueron aplicados en relación a el desarrollo con el motor de Unity, teniendo el propio motor una metodología y funciones específicas a esta.

# CAPÍTULO 8
# Trabajo futuro

Si bien el objetivo del TFG ha sido completado satisfactoriamente, debido a las limitaciones personales y temporales existen diferentes campos para expandir el desarrollo que no han podido ser aplicados.

Por un lado, se podría haber extendido la funcionalidad del generador de terreno, permitiendo al usuario crear las propias configuraciones de señales de ruido desde fuera del código fuente. En la misma línea, la posibilidad de cambiar la operación a voluntad también podría ser un aspecto posible a implementar. Con esto el terreno sería completamente modificable, pudiendo generar cualquier configuración deseada dadas las posibilidades del mismo. Otro acercamiento posible a desarrollar es la idea de crear formaciones como ríos, generando un agente que cree el curso del mismo dadas las diferencias en altura del terreno.

Para extender la funcionalidad del generador de construcciones el acercamiento más sencillo sería extender la cantidad de estrategias actuales, para generar los mismos edificios de diferentes maneras. Las ampliaciones podrían incluir el uso de diferentes tipos de suelos de dimensiones diferentes. Una ampliación que requeriría de una gran refactorización y de mucho trabajo sería la posibilidad de para unos elementos establecidos, aceptar una gramática nueva con los mismos para generar las construcciones en base a ella.

En referencia al generador de combinación, un punto muy grande que se abre al haberlo desarrollado es la posibilidad de conectar diferentes regiones de construcciones. Dados los resultados de las pruebas se puede apreciar que las construcciones se generan en focos localizados. Conociendo este dato se podría tratar de unir estos focos mediante caminos dibujados en el terreno, intentando esquivar altas montañas para conectar estas supuestas localidades.

En cuanto a optimización y rendimiento, sería posible generar aun más construcciones sobre el terreno si se pudiera simplificar los modelos provistos para reducir la cantidad de triángulos que los forman. A su vez, podría ser posible delegar la generación en diferentes pasos y no intentar generar todo en el mismo, posibilitando así generar un terreno de cualquier tamaño.

Llegando a este punto, está claro que existen múltiples posibilidades dentro del ámbito desarrollado del trabajo para la extensión del mismo, pero también es de conocer que la generación procedural es un campo muy extenso, sería posible, entre otras cosas, animar proceduralmente un personaje y hacerlo recorrer el terreno e interactuar con las casas. Así mismo, podría ser posible generar procedu-

ralmente las mismas partes que crean las construcciones, para así crear edificios completamente personalizados.

Es posible afirmar que la generación procedural es un campo amplio y extenso de la informática, con infinidad de posibilidades para crear todo aquello que se necesite.

# Bibliografía


[1] George S. Fishman. *Monte Carlo*. Springer New York, New York, NY, 1996.

[2] Mark Hendrikx, Sebastiaan Meijer, Joeri Van Der Velden, and Alexandru Iosup. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 9(1):1–22, February 2013.

[3] George Kelly and Hugh McCabe. Citygen: An interactive system for procedural city generation. 11 2007.

[4] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N. Yannakakis, and Julian Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37, January 2021.

[5] Daniel D. McCracken and Edwin D. Reilly. *Backus-Naur form (BNF)*, page 129–131. John Wiley and Sons Ltd., GBR, 2003.

[6] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers on - SIGGRAPH '06*, page 614, Boston, Massachusetts, 2006. ACM Press.

[7] Anton Nijholt. *Context-free grammars: covers, normal forms, and parsing*. Number 93 in Lecture notes in computer science. Springer-Verlag, Berlin ; New York, 1980.

[8] Ken Perlin. Improving noise. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 681–682, San Antonio Texas, July 2002. ACM.

[9] Noor Shaker, Julian Togelius, and Mark J. Nelson. *Procedural Content Generation in Games*. Computational Synthesis and Creative Systems. Springer International Publishing, Cham, 2016.

[10] Gillian Smith. Understanding procedural content generation: a design-centric analysis of the role of PCG in games. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 917–926, Toronto Ontario Canada, April 2014. ACM.

# APÉNDICE A
# Objetivos de desarrollo sostenible

En cuanto a los objetivos de desarrollo sostenible ODS, es preciso hacer una mención a diferentes de ellos debido a su presencia en el TFG desarrollado. Dadas las características del propio trabajo, siendo este una extensión orientada a la generación procedural para el motor Unity con el objetivo de crear una herramienta para la comunidad, se pueden relacionar en mayor manera ciertos objetivos, alineados de manera coherente con el objetivo del trabajo.

El primer objetivo con el que es clara la relación es con el de "Trabajo decente y crecimiento económico". El objetivo del trabajo es crear una manera para cualquier clase de usuario que lo desee de facilitar el desarrollo de diferentes elementos mediante la generación procedural. Este objetivo incluye ampliamente la concepción de trabajo decente y crecimiento económico, por un lado, reduciendo la necesidad de trabajos de gran coste humano y repetitivos, a la vez que permitiendo un ahorro de recursos y, por lo tanto, mayor oportunidad para el crecimiento económico. Es por ello que se cumplen ambos puntos del objetivo, por un lado, evitando el trabajo que es automatizable y, gracias a esto, evitando un consumo de recursos permitiendo el crecimiento económico.

El siguiente objetivo relacionado con el trabajo desarrollado es el de "Industria, innovación e infraestructuras". Sobre este mismo, en el ámbito de innovación se pretende desarrollar una nueva herramienta sobre lo ya existente, así creando nuevas maneras de trabajar y generar dicho contenido. Es decir, el objetivo de innovar se basa en utilizando las tecnologías existentes crear otras más avanzadas mediante la combinación y extensión de las mismas, llegando así a los resultados presentados en el TFG. Ya no únicamente es una propia innovación el propio resultado del trabajo, sino que al estar planteado como una herramienta lo que se pretende es dar una posibilidad para una futura innovación, viendo que esta será abierta a todo usuario que desee interactuar con la misma.

Otro objetivo de desarrollo sostenible presente en el TFG sería el de "Producción y consumo responsables". Este objetivo es de gran relación con el trabajo realizado, dado la cercana relación de la generación procedural con el consumo eficiente. Para concretar, el uso de la generación procedural surge en la mayoría de los casos debido a una necesidad de realizar o crear elementos de características similares. Dada la funcionalidad de la misma, aplicar esta resulta en una reducción en el coste humano para realizar dichas tareas repetitivas. A su vez, al reducir el coste humano en todo este proceso, el coste físico de desarrollo se redu-

ce, al ser necesario menor cantidad de recursos computacionales, estos traduciéndose en electricidad y horas humanas en este caso. De esta manera, la aplicación de la generación procedural implica un paso hacia un consumo responsable tanto con los recursos físicos, como los humanos.

Existe cierta relación entre el TFG realizado y otros objetivos de desarrollo sostenible, pero la mayor parte de ellos no tienen una clara relación, como por ejemplo "Fin de la pobreza", "Hambre cero" o "Vida submarina". Habría una pequeña relación con "Ciudades y comunidades sostenibles", dado que el ahorro de recursos, como en este caso es la electricidad, es un pequeño paso para alcanzar unas comunidades que operen de manera sostenible, pero el foco del mismo TFG no es este principalmente.

A continuación el grado de relación del trabajo con los objetivos de desarrollo sostenible en una tabla:

| Objetivos de Desarrollo Sostenibles | Alto | Medio | Bajo | No Procede |
|---|---|---|---|---|
| ODS 1. Fin de la pobreza. | | | | X |
| ODS 2. Hambre cero. | | | | X |
| ODS 3. Salud y bienestar. | | | | X |
| ODS 4. Educación de calidad. | | | | X |
| ODS 5. Igualdad de género. | | | | X |
| ODS 6. Agua limpia y saneamiento. | | | | X |
| ODS 7. Energía asequible y no contaminante. | | | | X |
| ODS 8. Trabajo decente y crecimiento económico. | X | | | |
| ODS 9. Industria, innovación e infraestructuras. | X | | | |
| ODS 10. Reducción de las desigualdades | | | | X |
| ODS 11. Ciudades y comunidades sostenibles | | | X | |
| ODS 12. Producción y consumo responsables. | X | | | |
| ODS 13. Acción por el clima. | | | | X |
| ODS 14. Vida submarina. | | | | X |
| ODS 15. Vida de ecosistemas terrestres. | | | | X |
| ODS 16. Paz, justicia e instituciones sólidas. | | | | X |
| ODS 17. Alianzas para lograr objetivos | | | | X |