Diseño y desarrollo de un armazón para la generación procedimental de terreno y vegetación en Unity

Design and development of a framework for procedural terrain and vegetation generation in Unity

Trabajo de Fin de Grado
Curso 2023–2024

Autores
Ignacio del Castillo Rubio
Javier Comas de Frutos
Sara Isabel García Moral
Javier Enrique Villegas Montelongo

Director
Federico Peinado Gil

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Diseño y desarrollo de un armazón para la generación procedimental de terreno y vegetación en Unity

Design and development of a framework for procedural terrain and vegetation generation in Unity

Trabajo de Fin de Grado en Desarrollo de Videojuegos
Departamento de Ingeniería de Software e Inteligencia Artificial

Autores
Ignacio del Castillo Rubio
Javier Comas de Frutos
Sara Isabel García Moral
Javier Enrique Villegas Montelongo

Tutor
Federico Peinado Gil

Convocatoria: *Junio 2024*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

27 de mayo de 2024

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento a todas las personas que han contribuido al desarrollo de este Trabajo de Fin de Grado.

En primer lugar, agradecemos a nuestro director de TFG, Federico Peinado Gil, por su orientación y apoyo a lo largo de este proceso.

A nuestras familias y amigos, por su constante apoyo y ánimo, indispensables para alcanzar esta meta.

Y por último a todos los compañeros desarrolladores que participaron en las pruebas de evaluación de nuestra herramienta.

Resumen

Es muy habitual encontrar blogs en Internet de desarrolladores de videojuegos independientes que en algún momento de su vida han pensado en hacer un proyecto que requiera la generación procedural de contenido, incluso que permita a cada jugador enfrentarse a un mundo virtual diferente cada vez.

Estos proyectos a menudo tienen dos factores en común. El primer factor es que suelen empezarse de cero, sin usar herramientas que ahorren trabajo a los desarrolladores, que tienden a programar sus propios generadores de terreno basados en ruido y sus propios posicionadores de vegetación. El segundo factor es que los productos finales son muy parecidos entre sí, siendo las únicas diferencias los recursos audiovisuales utilizados en cada proyecto.

En este trabajo proponemos una herramienta que busca ahorrar horas de trabajo a estos desarrolladores independientes interesados en usar la generación procedimental de terrenos y vegetación pero sin tener que construirse ellos mismos la infraestructura desde cero y pudiendo personalizar los resultados.

Esta herramienta permite al desarrollador generar mundos virtuales con varias características interesantes. En primer lugar, pueden utilizarse biomas adaptables a las características del terreno, es decir, a su altura (partes más altas en sistemas montañosos o partes más bajas en orillas), forma o texturizado. En segundo lugar, hay un sistema de generación de localizaciones y objetos relevantes para el juego. En tercer lugar, hay un posicionador de vegetación según biomas para dar vida al mundo generado. En cuarto lugar, es posible decidir si se usan texturas corrientes sobre un terreno continuo y suavizado o texturas planas sobre un terreno formado por cubos. En quinto y último lugar, cuenta con optimizaciones para mejorar el rendimiento en ejecución tanto del terreno como de la vegetación generada.

El funcionamiento se ha validado mediante la realización de pruebas en el laboratorio y también con usuarios reales de la herramienta.

Palabras clave

Creatividad Computacional, Generación Procedimental de Contenido, Videojuego, Unity, Bioma, Ruido, Altura.

Abstract

It is very common to find blogs on the Internet from independent game developers who at some point in their lives have thought about doing a project that requires procedural generation of content, even allowing each player to face a different virtual world each time.

These projects often have two factors in common. The first factor is that they are often started from scratch, without the use of labour-saving tools for developers, who tend to program their own noise-based terrain generators and their own vegetation positioners. The second factor is that the final products are very similar to each other, the only differences being the audiovisual resources used in each project.

In this paper we propose a tool that aims to save hours of work for those independent developers interested in using procedural generation of terrain and vegetation but without having to build the infrastructure themselves from scratch and being able to customise the results.

This tool allows the developer to generate virtual worlds with several interesting features. Firstly, biomes can be used that are adaptable to the terrain characteristics, i.e. height (higher parts in mountainous systems or lower parts on shores), shape or texturing. Secondly, there is a system for generating game-relevant locations and objects. Thirdly, there is a biome-based vegetation positioner to bring the generated world to life. Fourthly, it is possible to decide whether to use ordinary textures on a smooth, continuous terrain or flat textures on a terrain made up of cubes. Fifthly and finally, it has optimisations to improve the running performance of both the terrain and the generated vegetation.

The performance has been validated by testing in the lab and also with real users of the tool.

Translated with DeepL.com (free version)

Keywords

Computational Creativity, Procedural Content Generation, Video Game, Unity, Biome, Noise, Height.

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La generación procedimental (también llamada por procedimientos o con el anglicismo *procedural*) es el método de creación de contenido usando algoritmos en lugar de hacerlo de forma manual. Dependiendo del contexto en el que se utilice su finalidad es diferente. Al ser este un Trabajo de Fin de Grado (TFG) perteneciente al Grado en Desarrollo de Videojuegos, nuestro contexto son los videojuegos. La aplicación principal de esta técnica en este contexto es la creación de contenido como escenarios, misiones u objetos del juego.

Los algoritmos comúnmente usados para llevar a cabo este tipo de técnicas suelen estar basados en nodos, en autómatas celulares, o en ramificación y poda. Este tipo de algoritmos suelen ser recursivos (también llamados recurrentes). También son muy populares los algoritmos de ruido, que producen patrones pseudoaleatorios coherentes y suaves, proporcionando una buena base para modelar variaciones naturales, por ejemplo, en la topografía (ej. ruido de Perlin y ruido de Simplex).

El enfoque procedimental reduce considerablemente el tiempo de generación de un mundo virtual ya que el proceso para dar forma+ al mapa y colocar elementos como los árboles se automatiza por completo. También aumenta la variabilidad de los mundos y su aleatoriedad, lo que puede incrementar su realismo. Además si se realiza en tiempo de ejecución, proporciona experiencias únicas a cada jugador.

Es especialmente útil en videojuegos de mundo abierto donde la exploración es una parte integral de la experiencia del jugador. Cabe destacar que este tipo de juegos ha cobrado especial importancia en la última década debido al gran éxito de títulos pertenecientes a este género como *Grand Theft Auto V* Rockstar North (2013), *Red Dead Redemption 2* Rockstar Studios (2018) y *The Legend of Zelda: Breath of the Wild* Nintendo EPD (2017). Todos ellos se construyen sobre inmensos mundos virtuales con grandes posibilidades para la exploración e interacción.

1.1. Motivación

Muchos desarrolladores independientes (o *indies*, en inglés) que utilizan Unity no disponen de una herramienta potente que les permita generar terrenos de forma procedimental y completamente personalizable. En muchos casos las herramientas existentes en la Unity Asset Store son de pago y permiten crear tipos de mundos muy concretos y en general bastante pobres.

Muchos desarrolladores acaban creando sus propios algoritmos y estructuras de datos para generar los escenarios de sus juegos. Recurren a los mismos recursos de formación y, como resultado, los resultados suelen parecerse mucho entre sí.

1.1.1. Propósito

El propósito de este trabajo es contribuir a que existan herramientas de desarrollo que faciliten la generación procedimental de terrenos y vegetación en videojuegos en Unity. Se busca reducir la carga de trabajo e inversión de esfuerzo que los pequeños de desarrolladores necesitan llevar a cabo para construir sus propios generadores procedimentales de mundos virtuales.

El perfil de nuestros potenciales clientes es este, el desarrollador independiente interesado en mundos virtuales generados procedimentalmente. Es cierto que actualmente ya existen herramientas para la creación de terrenos, sin embargo, en el mercado apenas existen herramientas que creen mundos virtuales complejos, variados, con biomas y que sean altamente personalizables de una manera directa y sencilla. Por este motivo hemos optado por crear una herramienta que permite generar terrenos y vegetación de forma procedimental usando un editor, un conjunto de prefabs y mapas de alturas entre otros recursos básicos.

La herramienta se constituye como una extensión del entorno de desarrollo de videojuegos más utilizado hoy en día, que es *Unity* Technologies (2024). A nivel mundial un 73 % de estudios de desarrollo de menos de 50 empleados lo utilizan y concretamente en España, un 85 % de las empresas lo utiliza frente al segundo motor más utilizado, que es Unreal Engine, con un 23 %, según el *Libro Blanco de los Videojuegos.* de Empresas Desarrolladoras de Videojuegos y Software de Entretenimiento) (2022)

Las principales funciones que buscamos con este armazón propuesto son la sencillez y variabilidad. Para ilustrar las posibilidades de este armazón, una vez desarrollado, creamos varios prototipos de juego utilizando el software propuesto.

1.1.2. Alcance

El panorama actual de las herramientas para la creación de mundos procedimentales presenta una complejidad considerable. Se requiere un exhaus-

tivo análisis por parte de un desarrollador independiente para comprender a fondo el funcionamiento y la aplicación efectiva de estas herramientas. El objetivo principal de este proyecto radica en ofrecer una solución que permita a estos desarrolladores evitar ese trabajo de comprensión técnica de las herramientas en cuestión.

Nuestra propuesta se fundamenta en la minimización de esta carga de trabajo, buscando proporcionar una interfaz simple y funcionalidades claras que permitan al desarrollador independiente comenzar a generar mundos procedimentalmente con una inversión mínima de esfuerzo. Facilitando la integración de todas las funcionalidades requeridas para el desarrollo de su juego, ofrecemos una experiencia rápida y directa. Estas funcionalidades en la generación del mundo virtual se han ajustado meticulosamente para asegurar armonía y cohesión.

1.2. Asignaturas relacionadas

A continuación hacemos una descripción de las relaciones de este TFG con respecto a diferentes asignaturas cursadas durante la carrera.

La utilización de diferentes algoritmos de Inteligencia Artificial (IA) para la creación y modelado del terreno como el ruido de Perlin, hace que la asignatura de Inteligencia Artificial para Videojuegos (IAV) sea la más estrechamente relacionada con este trabajo. Estos algoritmos se basan en la recursividad por lo que esta relacionada con la asignatura de Matemáticas Discretas (MD) también.

El diseño tanto de la herramienta como del prototipo de ejemplo, así como el desarrollo de documentos de diseño (GDD, del inglés Game Design Document) se apoya en la asignatura de Diseño de Videojuegos (DV).

La producción del proyecto, organización y gestión del repositorio y tareas se basan en Metodologías Ágiles de Producción (MAP).

La asignatura Motores de Videojuegos (MOT) está también relacionada con el proyecto por el uso continuado del entorno de desarrollo de Unity.

El desarrollo de nuestra herramienta ha sido significativamente influenciado por la comprensión y aplicación práctica adquirida a través de asignaturas como Estructura de Datos y Algoritmos (EDA) y Métodos Algorítmicos en Resolución de Problemas (MARP). Estas asignaturas han sido pilares fundamentales en nuestra capacidad para abordar los desafíos algorítmicos y estructurales inherentes a este proyecto.

El Grado en Desarrollo de Videojuegos dispone de un listado de asignaturas relacionadas con la programación que, aunque no tengan una relación directa con la temática del trabajo, son imprescindibles, pues nos han proporcionado los conocimientos de programación y la habilidad para diseñar arquitecturas software y estructurar y resolver debidamente los problemas técnicos. Estas asignaturas son Fundamentos de la Programación (FP), Tec-

nología de la Programación de Videojuegos (TPV) e Informática Gráfica (IG).

Además, las diferentes asignaturas de Proyecto (I, II y III) y Prácticas en Empresa han aportado habilidades para trabajar en equipo y desarrollar un proyecto como este, más ambicioso y ocupando un mayor lapso temporal.

1.3. Estructura del trabajo

La estructura de este trabajo se compone de los siguientes apartados:

- **Capítulo 1: Introducción.** Se presenta la propuesta, el alcance, el propósito que tiene este trabajo y las asignaturas del grado relacionadas con este proyecto.
- **Capítulo 2: Estado de la cuestión.** Se revisa el estado de la cuestión en materia de IA usada dentro de los videojuegos o como técnica para ayudar en su desarrollo.
- **Capítulo 3: Objetivos y especificación.** Se presentan los objetivos principales del trabajo y se hace una especificación de todos los requisitos necesarios para poder desarrollar la herramienta propuesta.
- **Capítulo 4: Metodología.** Se explica la metodología usada para desarrollar el proyecto y qué herramientas concretas se han utilizado en el proceso.
- **Capítulo 5: Desarrollo.** Se realiza una explicación detallada del análisis, diseño e implementación de nuestro proyecto.
- **Capítulo 6: Resultados.** Se comienza con una explicación de las pruebas y los objetivos. Posteriormente, se comentan los resultados obtenidos de las pruebas con usuarios reales que hemos realizado para validar el funcionamiento de la herramienta y evaluar su utilidad para desarrolladores reales.
- **Capítulo 7: Conclusiones.** Se muestran las conclusiones generadas como producto de la comparación de la versión final de la herramienta con los objetivos definidos al principio, para validar si estos se han cumplido.

Capítulo 2

Estado de la cuestión

Este trabajo consiste en la creación de una herramienta en Unity para generar procedimental-mente terrenos con vegetación de una forma sencilla e intuitiva, accesible a cualquier desarrollador independiente. Estos terrenos tienen además la posibilidad de albergar distintos biomas.

Un bioma es una porción de terreno que comparte el clima, flora y fauna, teniendo este unas características definidas a partir de su vegetación y de las especies animales que predominan.

La herramienta esta basada en componentes, usará prefabs y mapas de alturas para dejar libertad al desarrollador a la hora de crear su propio mundo con su estética y elementos propios. De hecho, también tendrá un sistema de generación de elementos especiales para complementar el terreno.

2.1. Generación Procedimental

Al hablar de generación procedimental desde el punto de vista enfocado a videojuegos, podemos definir a esta como el conjunto de funciones y algoritmos que permiten crear mundos, niveles, texturas y en general cualquier tipo de contenido creado de manera automática a partir de unos parámetros establecidos por el desarrollador. Estos principalmente involucran el ruido y mapa de alturas.

2.1.1. Algoritmos de generación de Ruido

Existen gran variedad de algoritmos para la creación de terrenos y colocación de objetos, los más populares los podemos observar en la tabla (Archer, 2011) Tabla 2.1 :

2.1.2. Ruido

Hemos decidido usar el **ruido de Perlin** por las siguientes razones:

Ruido	Descripción
Ruido de Perlin	Este algoritmo fue desarrollado por *Ken Perlin* y permite generar patrones pseudoaleatorios que simulan un ruido de aspecto mas orgánico porque genera una secuencia naturalmente ordenada de números pseudoaleatorios es decir una secuencia de números aparentemente aleatorios pero ordenados de forma ascendente
Simplex Noise	Este algoritmo también fue ideado por Ken Perlin. Sin embargo, funciona dividiendo un plano en "simples", la forma más compacta posible para ese plano, y asigna un valor pseudoaleatorio a cada esquina. Las ventajas que tiene con respecto al ruido de Perlin es que se necesitan menos cálculos y por tanto opera mas rápido. (Generation, 2024)
Diamond-Square	Es un algoritmo que genera mapas de altura, tomando una textura de grises 2D como semilla y genera el mapa de altura a partir de ella. Además requiere estado, lo que tiene un coste en memoria potencialmente alto. (Sell, 2024)
Diagramas de Voronoi	Este se basa, en dividir una superficie en tantas áreas como puntos representativos haya, de manera que a cada punto se le asigna el área formada por todo lo que está más cerca del punto, mas que de ningún otro
Marching Squares	Este es un algoritmo mas inusual ya que suele usarse en informática gráfica para generar contornos para un campo escalar bidimensional (matriz rectangular de valores numéricos individuales). Sin embargo, también puede usarse para la generación de terrenos.

Tabla 2.1: Tabla de algoritmos de ruido

- En Unity ya existe en la librería Math el método para generar un valor de ruido de Perlin a partir de un plano 2D. Esto es importante porque permite al desarrollador centrarse en otras cuestiones de la generación procedimental.

- Mayor control con el ajuste de los parámetros que generan el ruido.

- Mayor escalabilidad desde el punto de vista de crear mapas grandes a pequeños.

- Mayor naturalidad ya que genera terrenos con un aspecto mucho mas orgánico porque genera una secuencia naturalmente ordenada de números pseudoaleatorios.

2.1.3. Mapas de alturas

Para la crear del mapa de alturas, nos basaremos en el pseudocódigo del manual de Millington (Millington y Funge, 2009) de la sección 8.3 "Landscape Generation", donde se describen técnicas que implican la creación de paisajes y todo lo que con lleva (ruido, octavas, escalas, procesos geológicos, etc.). Destacan los apartados 8.3.1 "Modifiers and Height-Map", 8.3.2 "Noise" y 8.3.3 "PerlinNoise", donde se explica la generación de mundos mediante la aplicación de modificadores.

2.2. Generación de Terreno

En esta sección se explican todos los elementos relacionados con la generación física del terreno, es decir, las mallas y sus niveles de detalle, su aspecto visual modificado con shaders y los biomas que contiene con sus propias características.

2.2.1. Biomas

Entendemos por biomas como las áreas de un terreno que tienen distintas propiedades entre sí, como por ejemplo un bosque y un desierto, siendo estos dos biomas diferentes dada la diferencia de flora y humedad entre ellos, utilizado en generadores procedimentales de terreno para dar más variedad al mundo.
Cada bioma suele tener atributos propios que permiten alterar distintas características de su zona del mundo, como la flora y/o la fauna, el tipo de terreno, su textura, la altura máxima y mínima, temperatura, humedad, etc.

2.2.2. Generación de mallas

Una malla se compone de triángulos organizados en un espacio 3D para crear la impresión de un objeto sólido. Un triángulo está definido por sus tres puntos de las esquinas o vértices.(docs.unity3d.com, 2021)

Lo primero que hay que tener en cuenta es que Unity tiene una limitación en la creación de vértices por malla de 65534 vértices por malla. De esta forma para la organización del mapa el cual va a tener un determinado tamaño, no se puede generar todo el mapa en una misma malla, ni mucho menos es óptimo.

Un chunk es un objeto software que contiene una porción del mapa, es decir, la malla de esa porción y los objetos que se encuentran en esa porción. El tamaño de esa porción se calcula a partir del tamaño de mapa establecido por el usuario.

En escenarios que suelen contener una gran cantidad de elementos y mallas con relativa complejidad (medida según el número de polígonos que las construyen, a mayor número de polígonos, mayor complejidad) es común el uso de LODs (Levels of Details) o niveles de detalle en español.

Los LODs definen, como su nombre indica, en nivel de detalle o complejidad de las mallas 3D. Para conseguir minimizar esa complejidad se lleva a cabo un proceso que reduce el número de triángulos de la malla. Para construir una malla en 3D es necesario construirla a base del uso del polígono con menor número de vértices, el cual es tres (el triángulo). Para reducir su cantidad se aumenta el tamaño de esos triángulos. La malla al tener menos polígonos es menos costosa de renderizar y pierde detalle, convirtiéndose en una figura más simple. Estos niveles de detalle suelen depender de la distancia de la cámara a la malla. A más cercanía, más detalle, a más distancia, menos complejidad de la malla. De hecho, existe un punto en el que la malla no se renderiza. Los objetos suelen tener de 2 a 4 niveles de detalle. Por tanto se deduce que se debe modelar o generar una malla distinta, es decir, con un número de polígonos distinto, que se asigna a cada nivel.

2.2.3. Terreno infinito

Endless Terrain se podría traducir literalmente como terreno infinito o sin fin. En el contexto de la generación procedimental es un algoritmo que se utiliza para generar un terreno procedimentalmente y que además sea infinito. Para que computacionalmente sea posible, consiste en crear únicamente los bloques o subsecciones de terreno sólo alrededor de la posición donde se suele encontrar la cámara o el jugador. Según este se vaya moviendo, se generarán los bloques de alrededor y los que se encuentran lo suficientemente alejados dejarán de ser visibles.

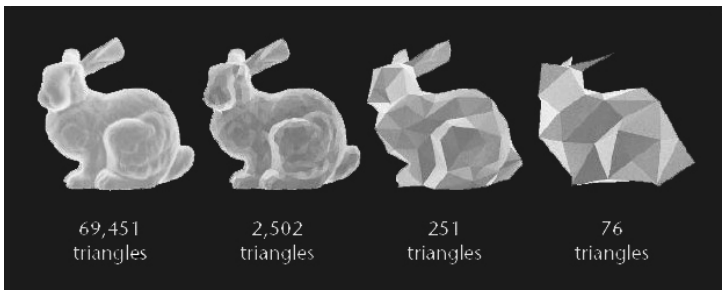
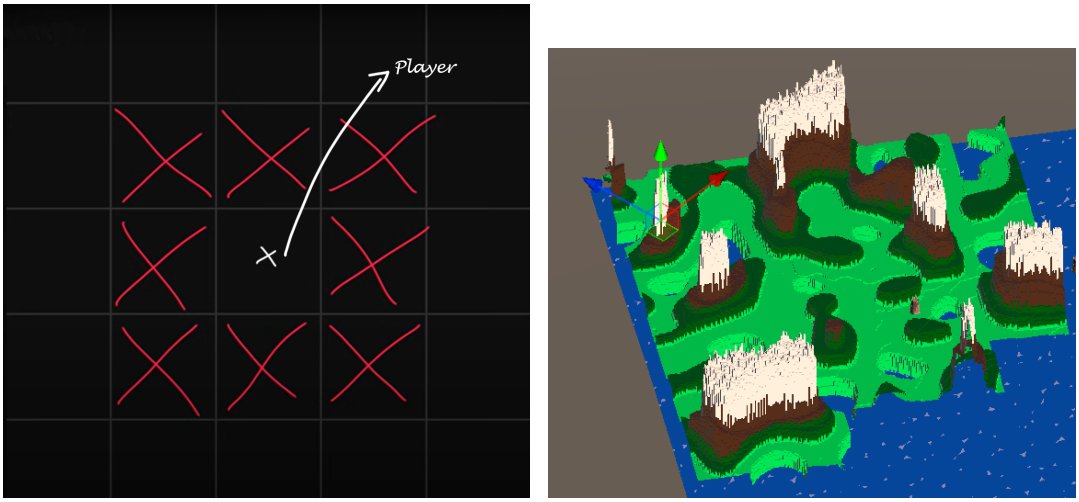


Figura 2.1: Ejemplo de LOD (LOD – Level of Detail, 2012) (https://hydrogen2014imac.wordpress.com/2012/10/20/lod-level-of-detail/)

Sin embargo el terreno no tiene porque ser infinito. Este algoritmo puede usarse para reducir el estrés de la CPU y aumentar el rendimiento, renderizando solo los bloques de terreno alrededor de un objeto que en muchos casos será el jugador. Se puede ver una representación esquemática en la siguiente imagen: 2.2a). En esta figura se ve una cuadricula que representa los bloques de terreno. Las cruces rojas representan los bloques de tierra que se van a renderizar. Las cuadriculas sin nada serán bloques no visibles.



(a) Esquema de Endless Terrain (b) Ejemplo

Figura 2.2: Imágenes del funcionamiento de Endless Terrain

2.2.4. Shaders

Un *'shader'* (sombreador en español) es un programa informático que realiza cálculos gráficos escrito en un lenguaje que se puede compilar de forma independiente y en tiempo real. El término 'shader'proviene del inglés 'shade'(sombra en español). Típicamente se ejecutan en la GPU (Unidad de Procesamiento Gráfico). Principalmente se utilizan para decorar las superficies de los modelos 3D en los videojuegos. No obstante tienen otras funciones como realizar transformaciones de vértices o coloreado de píxeles, entre otras muchas posibilidades, con el propósito de crear efectos especiales (niebla, fuego, etc.), reflejos, refracciones, o simulación de materiales de todo tipo.

Los 'shaders'toman la información de la geometría de los objetos 3D, como su posición, forma y textura, y la combinan con datos de iluminación, sombreado, cámaras y otros parámetros para calcular el color final de cada píxel en la pantalla.

Para su programación, los 'shadersútilizan lenguajes específicos de alto nivel independientes del hardware sobre el que se ejecuten.

Existen diferentes tipos de shaders, cada uno con una función específica en el proceso de renderización, como se puede ver en Tabla 2.2.

Fuente: (formacionprofesional.net)

Cuando hablamos de shaders a menudo aparecen texturas 2D. Esto implica hablar de las UVs, estas son coordenadas en 2 dimensiones, U equivale al eje X y V equivale a eje Y. Estas coordenadas se utilizan para adecuar la posición 3D en el espacio de un vértice a un plano bidimensional. Esto permite la asignación de una textura 2D en una malla 3D. Para ello, usualmente se asocia un par de coordenadas UV (2D) a un vértice de la malla. Esto determina cómo se coloca una textura 2D sobre una superficie poligonal.

2.3. Generación de Elementos

En esta sección se enumeran los elementos que pueden generarse en un terreno creado de forma procedimental.

2.3.1. Vegetación

El objetivo del sistema de vegetación es la generación de objetos, mas enfocada a vegetación entre otros dentro del entorno generado, mejorando así la inmersión y la estética del entorno, dando así la oportunidad de crear infinidad de juegos con distintas ambientaciones.

Normalmente a día de hoy se entiende por "vegetación" todo objeto que queremos instanciar por el mundo o por parte de el. Los árboles y arbustos

Tipo	Descripción
Vertex Shader	Se encarga de transformar los vértices de los objetos en el espacio 3D. Realiza operaciones matemáticas para calcular nuevas posiciones, rotaciones, escalas y deformaciones de los vértices en función de animaciones, interacciones del usuario u otros parámetros. El vertex shader es fundamental para lograr movimientos suaves y realistas de los objetos en pantalla
Pixel Shader (Fragment Shader)	Se ejecuta en cada píxel de la pantalla y determina el color final de dicho píxel. Aquí es donde se aplican los cálculos de iluminación, sombreado, texturas y efectos visuales. Es responsable de los detalles más finos de la imagen, como la apariencia de la superficie de un objeto, la simulación de materiales y la generación de efectos especiales
Geometry Shader	Permite crear y manipular la geometría de los objetos en tiempo real. Puede generar nuevas formas, duplicar objetos, suavizar bordes y realizar otras transformaciones geométricas. Es especialmente útil para generar partículas, explosiones, deformaciones dinámicas y otros efectos que requieren cambios en la estructura de los objetos
Tessellation Shader	Se utiliza para mejorar la calidad y el nivel de detalle de las superficies 3D. Permite subdividir una malla de polígonos en fragmentos más pequeños, lo que proporciona una mayor precisión en la representación de curvas y superficies complejas. Es esencial para lograr modelos realistas en escenas detalladas y paisajes naturales

Tabla 2.2: Tabla de tipos de shaders

suelen ser el objeto típico a instanciar.

2.3.2. Puntos de interés

En el contexto de un videojuego, los puntos de interés son ubicaciones concretas dentro del propio mundo que pueden resultar interesantes, útiles o tener algún tipo de relevancia para el jugador.

Según su interés pueden ser:

- **Interés en la trama**. Lugares clave como mazmorras, campamentos enemigos, etc.
- **Interés en la jugabilidad**. Lugares que influyen en la jugabilidad, como tiendas, puntos de recolección de recursos o sanación, etc.

Los puntos de interés tienen como función principal atraer la atención del jugador. Esto puede deberse a que se busca fomentar la exploración del mundo del juego (sobretodo en mundos abiertos o de gran tamaño), simplemente pura decoración o convertirse en una elección de diseño para guiar al jugador en su partida u ayudarle con la jugabilidad.

En un mundo generado procedimentalmente a menudo este recurso se utiliza para animar al jugador a explorar todo el terreno generado. Estos puntos no necesitan ser dotados de una funcionalidad, simplemente pueden ser decorativos. Esto no implica que los desarrolladores no los usen para intervenir en la jugabilidad del usuario.

Un buen ejemplo es el uso de puntos de interés en el juego *Red Dead Redemption 2*. Este juego posee un gran mapa abierto, que se puede recorrer por el jugador libremente sin tener que seguir ningún orden o camino concreto. Los puntos de interés en este caso aparecen reflejados en el mapa para destacarlos con más facilidad (véase en: Figura 5.6).). En este caso todos tienen alguna funcionalidad: tiendas, establos, misiones, misiones secundarias, eventos varios, etc.

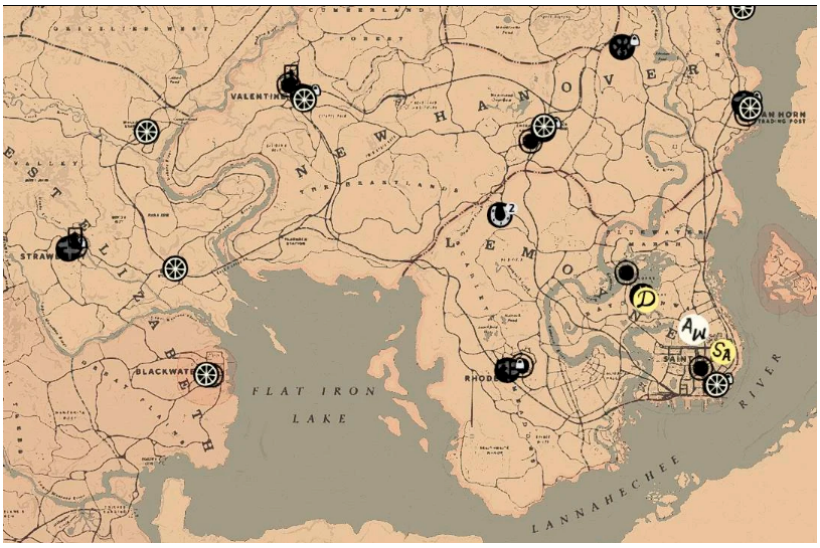


Figura 2.3: Mapa de Read Dead Redemption 2

2.4. Investigación

Como parte del proceso de desarrollo de nuestra herramienta, es importante comenzar realizando un estudio de mercado para conocer el estado de

la cuestión. Este análisis nos permitirá comprender a fondo el panorama actual en el segmento de herramientas disponibles para la creación de mundos procedurales.

Particularmente, nos enfocaremos en investigar y analizar las ofertas presentes en la Unity Asset Store, dado que Unity en sí no cuenta con una herramienta integrada para la creación de terrenos completamente procedurales. Además, exploraremos herramientas de Unreal Engine aunque no es el entorno de desarrollo que usamos en este trabajo para obtener una visión más completa del espectro de herramientas disponibles en la industria. Además de revisar herramientas, también hemos analizado varios videojuegos que destacan en la generación procedimental de su mundo.

2.4.1. Herramientas en el mercado

En esta sección vamos a analizar las herramientas que podemos encontrar actualmente tanto en el entorno de desarrollo de Unity como en UnrealEngine y alguna herramienta que destaque para la creación de contenido procedural.

2.4.1.1. Unity

Para Unity encontramos diversas herramientas, ya sea a través del propio motor o en la Assets Store, donde podemos encontrar diversos paquetes bastante complejos:

- **Unity Terrain**

  Unity tiene un objeto llamado Terrain un componente del mismo nombre que permite crear distintos terrenos. Este objeto consta de un plano cuya malla es modificable gracias a las diversas configuraciones que permiten ampliar la variedad de terrenos que se pueden generar y que ofrece su componente. Algunos ejemplos de estas son: tamaño y resolución del mapa, viento (solo afecta a los elementos colocados desde este componente) y colocación de objetos como vegetación, que reciben el nombre de "Tree". También posee otra sección de dibujado, con la que se pueden aplicar distintas texturas al terreno o pintarlas mediante un pincel. El pincel tiene distintas formas, tamaños u opacidades. También permite modelar el terreno a su paso, elevando o bajando los puntos por los que pasa con mayor o menor intensidad. La sección de "Tree" permite añadir todos los modelos que se requieran asociados a los objetos o vegetación que se va a colocar en el terreno. El proceso de colocación también funciona con un pincel.

  La desventaja de esta herramienta es que prácticamente la totalidad de la construcción del terreno es manual, no procedimental en sí. Además esto hace que el proceso de generación sea más laborioso y largo.

- **Mapmagic World Generator**

  Mapmagic World Generator es una herramienta que se encuentra en la Unity Asset Store, que se basa en una interfaz de scripting visual estructurada en nodos, con el propósito de determinar la lógica de creación del mundo. Usa algoritmos de Perlin Noise para generar los terrenos. Una vez están configurados estos nodos y conectados, se genera un mundo, que puede ser bastante grande, pero no permite una compleja erosión del terreno ya que tiene una por defecto. No obstante, es esencial tener en cuenta que, aunque esta puede ser una herramienta útil, también requiere tiempo para adquirir un flujo de trabajo eficaz y rápido para la creación de mundos detallados y realistas. No hay mucha documentación de la herramienta por lo que hay que ir descubriendo las cosas por cuenta propia. Este puede ser un proceso sumamente complejo, ya que requiere una comprensión exhaustiva de los diseñadores y cómo interactúan entre sí.

  Hay que tener en cuenta que lo dicho es relativo a la versión de prueba y es posible que la versión de pago cuente con una mejor estructura y documentación.

  [1] *Forum Unity Magic World Generator*

  AssetStore Magic World Generator (Pahunov, 2023)

- **World Creator**

  World Creator es una herramienta de generación de paisajes y terrenos 3D de manera procedimental en tiempo real, útil tanto para videojuegos, como para películas o simulaciones.

  Permite una amplia gama de posibilidades en la generación de terrenos debido a todos los parámetros y opciones que presenta. Estas son: mapas de calor que permiten previsualizar las acciones que se van a llevar a cabo, máscaras y filtros que modifican el terreno una vez generado (esto incluye sedimentación y erosión), creación de ríos y carreteras, colocación de objetos por todo el terreno incluso dependiendo de ciertas texturas, entre otras funciones.

  No obstante, presenta diversos problemas. Debido a la flexibilidad que presenta, su complejidad aumenta en consonancia. Sin embargo, posee muy pocas opciones a la hora de generar el terreno, simplemente se debe elegir una semilla y el terreno se genera de forma totalmente aleatoria sin dar opción al usuario de configurarlo a su gusto o poder previsualizarlo antes de confirmar la generación. Para tener más control se deben añadir máscaras y filtros posteriormente. El sistema de texturizado es complejo y se aplica al terreno como baldosas, sin opción de escalarlo, lo que resta realismo al texturizado. Además todas las

[1] *Forum Unity Magic World Generator*

texturas se mezclan entre ellas automáticamente, quitando al usuario control sobre ellas, sobretodo si quiere un mundo poco realista y más estilo *cartoon*. No hemos de olvidar tampoco que la herramienta no es gratuita y eso es un inconveniente para que los desarrolladores pequeños la prueben. De hecho, para poder usarla al completo es necesario obtener la versión profesional, que tiene un precio bastante elevado.

Para dominar esta herramienta hay que tener en cuenta la curva de aprendizaje, pues requiere bastante tiempo y práctica el llegar a comprender y aprovechar al máximo todas sus funciones.

[2] *API World Creator*

AssetStore World Creator (Unity Asset Store, 2023d)

- **TerraWorld 2023**

  Al igual que las otras herramientas anteriores, TerraWorld está diseñada para la creación de terrenos en 3D. Se trata de un generador de mundos 3D reales de alto rendimiento con un avanzado renderizador de GPU y eventos controlados por la cámara para el renderizado y la s elección inteligente de objetos. Permite generar grandes niveles 3D muy detallados, ya sea para juegos o simulaciones. Este se basa en un sistema de nodos, "a modo de scripting visual", para definir el entorno deseado. Estos nodos permiten definir la erosión, los biomas y la vegetación a aplicar. En cuanto al estilo de los gráficos se puede cambiar de un aspecto muy realista a low-poly, entre otros.

  Los terrenos que genera son capaces de adaptarse a diferentes escalas. Además, este paquete mediante los eventos controlados por cámara y la renderización con GPU avanzada prioriza la optimización, permitiendo que los terrenos creados ofrezcan un rendimiento óptimo en tiempo real.

  No obstante, TerraWorld presenta algunos obstáculos. A pesar de ser versátil, es difícil lograr un control artístico preciso en relación con los detalles del entorno. Además, debido a su interfaz basada en nodos y los algoritmos predefinidos, los diseños pueden llegar a carecer de originalidad. Esto implica que aprender a utilizar TerraWorld puede requerir bastante tiempo y práctica debido a la complejidad de sus nodos y ajustes para poder aprovechar toda la funcionalidad que ofrece.

  [3] *Forms TerraWorld*

  AssetStore TerraWorld (Unity Asset Store, 2023c)

- **Gaia Pro 2021**

  Al igual que las otras herramientas anteriores, es una herramienta de creación de terrenos y paisajes en 3D, similar a *World Creator* pero con

[2] *API World Creator*
[3] *Forms TerraWorld*

alguna particularidad diferenciadora. Además es compatible con varias versiones de Unity, URP y HDRP. Y cabe mencionar que incorpora el componente Terrain de Unity anteriormente mencionado.

La principal diferencia de Gaia Pro es que permite generar terrenos de dos formas distintas: una más rápida y aleatoria, y otra más lenta que permite un modelado (puede ser manual o semi manual) del terreno en tiempo real. En ambas formas el usuario puede definir el tamaño del mapa, su resolución y el bioma que se genera junto sus *spawners* (generadores de pueblos, árboles, etc.).

Si optamos por la forma lenta y compleja se puede configurar el ruido, su escala y altura, las texturas del bioma, los objetos que se generan y en qué textura, y la creación de ríos o montañas mediante *stamps*(modelos de un terreno ya predefinidos que se pueden colocar y adaptar al terreno que se está creando) o mediante un modelado manual con el pincel del componente Terrain, entre otras cosas. Es bastante flexible y admite infinidad de posibilidades.

Sin embargo, al igual que otras herramientas, Gaia Pro presenta desafíos en cuanto al control artístico preciso sobre los detalles del terreno, dificultando obtener resultados específicos para ciertas estéticas, sobretodo las menos realistas. Otra desventaja importante es que sólo permite crear un bioma por terreno. Y la forma de generar y posicionar objetos ya esta predefinida en scripts internos. Por ultimo, aprender a utilizar Gaia Pro puede requerir bastante tiempo y práctica para dominar todas sus funciones y ajustes, similar a otras herramientas de creación de terrenos, además de que su precio es uno de los más elevados del mercado.

[4]*API Gaia Pro 2021*

AssetStore Gaia Pro 2021 (Unity Asset Store, 2021)

- **TerraForge**

  TerraForge es una herramienta especializada en la creación de terrenos y paisajes. Esto lo hace mediante diferentes algoritmos de ruido, principalmente *Ruido de Perlin* y *OpenSimplex*. Es sencilla de usar, y permite crear tanto paisajes realistas, como otros más cartoon. Permite aplicar texturas por alturas e incorpora la herramienta Terrain de Unity también.

  En cuanto a la eficiencia, destaca por su capacidad para generar terrenos de manera rápida y eficiente.

  Un aspecto bastante a destacar es que la herramienta posee gran cantidad de tutoriales y documentación, a pesar de que es todavía una

[4]*API Gaia Pro 2021*

versión beta y una comunidad. La documentación de esta es mucho mas extensa y detallada que la de las otras herramientas.

La desventaja en esta herramienta es que no permite colocar vegetación de forma procedimental así como otros elementos, como por ejemplo puntos de interés. Se centra únicamente en la creación del terreno. Además no se encuentra de forma gratuita.

[5] *GitHub Docs TerraForge*

AssetStore TerraForge (Unity Asset Store, 2023b)

2.4.1.2. Unreal Engine

En Unreal Engine existen diversas herramientas para generar terrenos de forma procedural, ya puede ser con la herramienta Landscape del propio motor o las que se pueden encontrar en su Marketplace, aunque la mayoría de estas son para generación de objetos o materiales procedurales, no terrenos. Cabe destacar que todas las herramientas de la tienda dependen de Landscape de Unreal, ya que no generan mallas, si no Landscapes.

- **Landscape:** Herramienta de modelado de terreno bastante completa, integrada en Unreal Engine, que permite crear muchas combinaciones distintas a la hora de producir paisajes. El usuario es el que decide al detalle como modelar un área y dónde colocar los elementos, los materiales y las texturas que aplicar. Funciona como un pincel de área. Primero se debe seleccionar el tipo que acción que queremos llevar a cabo, como por ejemplo, suavizar o aplanar. También podemos seleccionar la intensidad y el tamaño del pincel. Tras esto, cada zona por la que pasemos el recién configurado pincel, cambiará y se aplicará allí la acción seleccionada. Todo esto se divide en 3 modos: gestionar, en este modo se puede crear nuevos terrenos o modificar algunos existentes, así como sus componentes; esculpir, este modo permite modelar las alturas del mapa de distintas formas (erosionar, elevar, aplanar...) mediante un pincel; y pintar, que permite modificar la apariencia del terreno aplicando de forma selectiva capas de materiales.

- **Foliage:** Herramienta integrada en Unreal Engine que permite colocar diversos recursos preseleccionados previamente por el usuario utilizando un pincel sobre una superficie que actúa como terreno. Todos esos recursos se adaptan a la forma del terreno. Esta herramienta no modifica el terreno en sí, únicamente permite colocar elementos en uno ya existente. La colocación de estos recursos admite una amplia configuración como: densidad, rotación, aleatoriedad, altura y escala entre otras cosas. Esta colocación se hace simulando un área del pincel, cuyo

[5] *GitHub Docs TerraForge*

tamaño es configurable, con el ratón. También existen otras opciones de colocación como rellenar un área determinada, colocar un solo elemento o borrar los elementos de algunas zonas.

- **Unreal Engine PCG:** Nueva herramienta en forma de plugin disponible en Unreal Engine que permite colocar elementos/modelos preseleccionados por el usuario previamente (árboles, rocas, etc.). Admite más variaciones y es más compleja que las preexistentes. Entre sus características vemos que permite hacer una previsualización de la colocación que van a tener los objetos representados por cubos con colores que van del blanco al negro. La colocación de objetos se hace dentro de un volumen con forma cúbica cuyas dimensiones son configurables. No se requieren conocimientos de programación textual porque funciona con programación visual mediante nodos. Además de todo esto tiene opciones para poder configurar la altura, densidad, rotación, etc. de los elementos, y evita solapamientos entre los objetos colocados (Epic Games, 2023). Permite también colocar *splines* con distintas formas que eliminan los objetos generados que colisionen con ellas, lo cual amplía las posibilidades de personalización de los escenarios. Se pueden crear caminos, carreteras, lagos o claros en los bosques, por ejemplo.

  Todos los elementos generados se adaptan al terreno subyacente.

- **Procedural Landscape Generator:**

  Esta herramienta se encuentra en el Unreal Marketplace, y permite generar terrenos de forma procedimental usando los blueprints de Unreal, que podrían definirse como programación visual. Utiliza diversos algoritmos de ruido como Perlin y Voronoi.
  Además también permite la generación de vegetación mediante ruido, generación de un plano de agua debajo del cual se elimina la vegetación y la instanciación de actores de Unreal por el terreno y todo de forma procedural.

  Posee una buena y extensa documentación.

  No obstante, sin su documentación la dificultad del uso de esta herramienta es mayor que todas las anteriores nombradas. Además, el usuario debe tener conocimientos de programación ya que no se usa el editor en ningún momento. Para continuar, no permite la creación de distintos biomas, todo el terreno es un conjunto. Y su precio es elevado.

- **Ultimate Landscape Generator:**

  En esta caso hablamos de un plugin de Unreal que permite crear y configurar distintos mapas de ruido y de alturas y exportarlos a Unreal por capas. Estas capas de distintos mapas se pueden superponer y crear configuraciones muy variadas que más tarde serán usadas por la herramienta Landscape para crear la malla correspondiente.

La documentación se basa en unos tutoriales en vídeo bastante ricos en contenido y explicaciones.

Las desventajas de esta herramienta en cuanto a la generación de mundos se refiere, se pueden enumerar varias, como la carencia de generación de vegetación u objetos, la inexistencia de biomas y lo tedioso que resulta crear materiales en Unreal que se adapten al terreno. Para esta última acción es necesario usar Landscape.

No es una herramienta gratuita pero su precio es asequible y acorde con lo que ofrece.

2.4.1.3. SpeedTree

SpeedTree es un software que emplea algoritmos y reglas predefinidas para generar la geometría de árboles y vegetación en lugar de depender de modelos preexistentes. Este enfoque se basa en algoritmos que definen aspectos como la distribución de hojas, el crecimiento de las ramas y otros detalles, permitiendo crear y personalizar la apariencia de la vegetación de manera flexible y realista.

Por ello es una herramienta de modelado 3D excepcionalmente versátil, conocida por su amplia personalización. Permite a los usuarios ajustar una gran cantidad de parámetros, desde la forma de las ramas hasta la textura de las hojas, para crear una variedad casi ilimitada de árboles y plantas con aspecto natural.

A pesar de su capacidad para crear vegetación detallada, SpeedTree está diseñado para ser eficiente en términos de recursos computacionales. Una de las mayores fortalezas de este es su adaptabilidad a diferentes plataformas. Se integra sin problemas con varios programas de modelado 3D y motores de juegos, como por ejemplo Unity, Unreal Engine y CryEngine, lo que facilita su uso en una amplia gama de plataformas y proyectos.

Por último, pero no menos importante, SpeedTree es conocido por su realismo y calidad visual. Su capacidad para generar vegetación realista ha hecho que sea una herramienta fundamental en la creación de entornos virtuales inmersivos, ya sea en videojuegos, películas, simulaciones o visualizaciones arquitectónicas.

2.4.1.4. Houdini Engine

Herramienta relacionada con el campo del modelado y animación 3D que cubre varias áreas como el modelado, animación, *rigging*, sistemas de partículas, generación de terrenos, iluminación y texturizado entre otras. Lo interesante es que permite realizar estas acciones de forma procedimental, siendo especialmente relevantes para este trabajo las herramientas de generación de terrenos y modelado procedimental. Houdini se puede acoplar tanto

a *Unity* como a *Unreal Engine* para trabajar de forma integrada con ambos motores (Unity Asset Store, 2023a).

Las herramientas de modelado de Houdini hacen que sea más fácil crear una buena topología de superficie para su uso en cine, TV, juegos y VR; y se pueden utilizar de forma interactiva en la ventana gráfica o de forma procedimental en el editor. Se puede elegir entre una amplia variedad de polígonos, NURBS y herramientas de modelado de superficies de subdivisión, mientras que las técnicas de procedimiento se puede utilizar para mantener un historial de construcción robusta y crear modelos muy detallados.

El sistema de terrenos de Houdini permite estratificar y editar el terreno fácilmente con herramientas como el ruido procedimental, o pintar y enmascarar áreas directamente. Las operaciones son muy similares a la composición de imágenes, por lo que los artistas del terreno encontrarán familiar el flujo de trabajo de Houdini. El terreno puede también colisionar con simulaciones de fluidos, partículas, RBD/destrucción para simular la destrucción de terrenos y objetos, Pyro FX para simular efectos como fuego, humo y explosiones - sin convertir los campos de altura en geometría.

También existe la posibilidad de modificar las alturas del terreno generado aleatoriamente con ruido para crear biomas distinguibles entre sí. Por ejemplo, que los desiertos interpreten el mapa de alturas generado de forma que no haya mucha variación entre el punto más alto y el punto más bajo, y que la curva de edición del mapa de alturas sea muy suave. Mientras que en un bioma de montañas quieres exactamente lo opuesto, es decir, que se pronuncien mucho las alturas extremas, es decir una curva más abrupta.

En ambos casos se usa un sistema de nodos, algo bastante común en la generación procedimental. Los dos casos contemplan una amplia gama de parámetros para configurar los modelos o terrenos como la escala, posición, orientación, subdivisiones, patrones (con sus propias configuraciones), ruido (octavas, rugosidad, amplitud...). Y todo esto se incluye en su versión *indie*, que es su versión más asequible (SideFX, 2023).

2.4.2. Videojuegos con generación procedimental de contenido

Mostramos tan sólo algunos ejemplos de videojuegos donde encontramos generación procedimental de contenido, especialmente de terrenos y vegetación.

2.4.2.1. Minecraft

Minecraft es un juego de construcción y aventuras tridimensional que ha ganado una enorme popularidad desde su lanzamiento en 2011. Desarrollado por Mojang y posteriormente adquirido por Microsoft, este juego es conocido principalmente por su mundo abierto y generado procedimentalmente. Este

mundo está formado íntegramente por bloques cúbicos y está organizado en una serie de biomas, que incluyen desiertos, bosques, océanos y mucho más.

La generación procedimental utiliza un algoritmo de generación de terrenos que se basa en una semilla. La semilla es un número o una cadena de texto que se utiliza como entrada de este algoritmo pseudoaleatorio. Cada una genera un mundo único y predecible (si dos jugadores utilizan la misma semilla, generarán mundos idénticos o muy similares). Se generan distintos tipos de *chunks* (superficies de 16x16x256 bloques) los cuales pueden estar en los estados: **generados** (donde se encuentra el jugador y alrededores, manejan objetos, así que los que se encuentran aquí, desaparecer en pocos minutos), **lazy** (forman un marco alrededor de los generados, no manejan objetos, así que estos nunca despawnean) y **no generados** (el jugador todavía no ha pasado, no se renderizan, no tienen nada, cuando el jugador pase se convertirán en generados).

2.4.2.2. LEGO Fornite

Este juego ha salido recientemente a finales del año 2023 y ha sorprendido todos al utilizar generación procedimental de contenido para crear de manera dinámica entornos detallados. Es bastante similar al Minecraft en términos de generación del mundo. Este mapa tiene distintos biomas también, además de criaturas, tanto hostiles como amistosas.

Al igual que Minecraft se basa en una semilla y funciona de forma parecida. En cada mapa que se crea, interviene una selección de los biomas existentes. En cada bioma aparece una selección de criaturas coherentes con esa zona y que tienen sus propias variantes. Cabe destacar que hay reglas como que nada más iniciar el juego, el jugador siempre aparece en un bioma forestal o zona de hierba, pues son las más seguras.

(LEGO Group, 2023)

2.4.2.3. Otros

Hay ejemplos de otros juegos, quizá menos claros y menos representativos, que usan la generación procedimental en mayor o menor medida.

The Division (Ubisoft Massive, 2016) utiliza la generación procedimental en los interiores (disposición de los muebles) mientras que los edificios tienen unas reglas fijas.(Ubisoft, 2016)

Borderlands (Gearsoft Software, 2009) presumía de más de tres millones de armas gracias a un sistema que utilizaba la inteligencia artificial del juego para combinar los materiales y generar aleatoriamente tipos de armas.(Software, 2009)

Gran Turismo 5 (Polyphony Digital, 2010) implementó un generador de circuitos aleatorio. (Digital, 2010)

Red Dead Redemption (Rockstar San Diego, 2010) mostraba una

batería de misiones aleatorias cada vez que viajábamos de un punto a otro. (Diego, 2010)

Spore es un videojuego en el que se creaban microorganismos y se generan galaxias enteras donde crecer y evolucionar como ecosistema. La banda sonora, parte de Brian Eno en colaboración con Peter Chilvers, también utilizaba música procedimental.(Maxis, 2008)

Con **No Man's Sky** pasa algo parecido. Las galaxias, los planetas (tamaño y biomas), naves y otros objetos son creados de forma procedimental.(Games, 2016)

RoboBlitz (Naked Sky Entertainment, 2006) aplica texturas generadas procedimentalmente.(Entertainment, 2006)

Dwarf Fortress ofrece 2 opciones para generar el mundo, uno de forma automática, tan solo interactuando con un botón y unos pocos parámetros u otro camino más complejo en el que definir todos los parámetros existentes. Este mundo generado es único: altura, condiciones climáticas, temperatura ambiental, pueblos y NPC's e historias con sus propios conflictos bélicos, trasfondo histórico de generaciones, flora y fauna. Estos elementos son generados exclusivamente al comienzo de cada partida. (y Zach Adams, 2006)

Capítulo 3

Objetivos y especificación

En este capítulo se describen los objetivos del trabajo y la especificación de la herramienta que se pretende desarrollar.

3.1. Objetivos

En este apartado se expone una enumeración de los objetivos propuestos para ser cumplimentados durante el proyecto.

1. Diseño de un sistema que permita generar de manera procedimental un mundo 3D. Se podrá generar un terreno de estética realista, con mallas suavizadas y orgánicas, o un terreno compuesto a base de cubos.

2. Diseño de un sistema de *biomas* que permita distribuir regiones distintas sobre el terreno. Estos *biomas* se definen incluyendo sus características, como parámetros de ruido que determinan las características de terreno, alturas máxima y mínima y la probabilidad de que dicho bioma aparezca en el mundo.

3. Implementar un sistema intuitivo de *shaders* para personalizar los materiales a usar en cada *bioma*. Cada bioma se divide en capas, las cuales pueden tener materiales distintos, que se pueden mezclar mediante transiciones suaves.

4. Diseño de un sistema de *vegetación* que permita crear múltiples ejemplares de un objeto sobre el terreno generado, según sus *biomas*. El posicionamiento y otras formas de adaptación al entorno son configurables en la vegetación.

5. Optimización que permita manejar eficientemente mundos de gran tamaño, renderizando in-game sólo los chunks de terreno y la vegetación disponga de distintos mesh con menos vértices, pudiendo así cambiar

la malla a otra de menor resolución según la distancia visible, tanto de la vegetación como de los chunks.

6. Creación de uno o varios mundos de demostración, cada uno con distintos *biomas*, *vegetación*, materiales y puntos de interés.
7. Comprobar la facilidad de uso de la herramienta realizando pruebas con desarrolladores reales. Dichas pruebas consistirán en tratar de replicar paisajes reales mediante la herramienta.

3.2. Especificación

En esta sección se enumeran los requisitos necesarios para poder cumplimentar el total de los objetivos de la forma más detallada posible.

3.2.1. Flujo de trabajo

Nuestro objetivo es que la herramienta sea intuitiva, fácil de usar y de aprender.
El flujo de trabajo que esperamos es el siguiente:

- Crear los biomas base de su proyecto: Determinar qué biomas necesita el usuario para poder generar el mundo que desee, y crear los objetos de tipo bioma necesarios, con valores elegidos a ojo o probando a generar un mundo con solo ese bioma para ajustar más precisamente los valores.
- Crear la vegetación necesaria: Utilizando prefabs que haya creado el mismo usuario, hayamos ofrecido nosotros de prueba o si son externos, crear los objetos que formarán la vegetación de su mundo y colocarlos en los biomas correspondientes.
- Generación de un mundo al azar: Generar un mundo de prueba para ajustar valores de la transición de biomas, la semilla deseada, si se quiere que sea una isla, etc.
- Crear los Puntos de interés deseados: Crear los objetos de puntos de interés para el mundo, decidiendo cuántos pueden aparecer, a qué distancia tienen que estar el uno del otro, etc.
- Generar el mundo final: Ya con todo creado, generar el mapa final con los puntos de interés, e ir cambiando la semilla hasta encontrar la perfecta. Una vez se encuentra la semilla, el usuario puede mover los puntos de interés alrededor del mapa para ajustar su posición en caso de que sea necesario.

También será capaz el usuario de dejar los parámetros a su gusto y hacer que el mundo se genere durante la ejecución de su juego, que con sólo modificando la semilla conseguirá mundos variados, pero con los mismos atributos.

3.2.2. Demostraciones

En lo referente al objetivo de creación de una o varias demos de la herramienta, se han construido tres. Cada una con unas características distintas que hacen que se diferencien entre todas.
Primero se ha creado una carpeta llamada "Demos" en la raíz del repositorio donde se encuentra el proyecto. Esta carpeta contiene una réplica del proyecto de la herramienta pero con recursos extras extraídos de paquetes gratuitos de la Unity Asset Store. Debido al uso de estos paquetes se ha separado en una carpeta a parte, de esta forma se evita la mezcla con la herramienta principal.
Dentro de esta carpeta se encuentra en proyecto con las demos. Cada una se encuentra en una escena distinta, dentro de las tres existentes: Sample1, Sample2 y Sample3. Cada una tendrá sus propios puntos de interés, vegetación, biomas y decoraciones pertinentes, coherentes con el escenario.

- **Sample1**. Aspecto boscoso
- **Sample2**. Desierto
- **Sample3**. Montañas nevadas

3.2.3. Interfaz del editor

Prácticamente toda la herramienta se basa en la modificación de los valores de los distintos parámetros desde el propio editor. Es por ello que debemos procurar que la interfaz sea completa.

Todos lo parámetros a modificar pertenecerán a los distintos scripts que poseerá el objeto maestro que se encarga de la generación del mundo o terreno (Map Generator). Los campos que debe tener para poder crear un terreno altamente personalizable son:

- Tipo de terreno: Permite elegir si se quiere mostrar en la escena la previsualización del mundo en 2D o directamente el mapa en 3D. Esto permite diferenciar lo tipos de terrenos 3D como son el suavizado o el cúbico.
- Tamaño del mapa: Al ser un mapa cuadrado solo hace falta especificar las dimensiones del lado. Se pide en las unidades que utiliza Unity.
- Semilla: Cambiar la semilla significa cambiar la distribución del ruido y las alturas del mapa, permitiendo crear mapas distintos.
- Biomas: El usuario tiene la posibilidad de crear todos los biomas que quiera, y cada bioma generará su propio terreno, con la mayoría de ajustes pudiendo ser retocados por el usuario, como varios atributos

del ruido utilizado para generar el terreno, la altura máxima y mínima posibles del bioma, cuánto debe aparecer con respecto a otros biomas y una curva para determinar cómo debe el bioma interpretar los valores del mapa de ruido.
Además, damos la opción de ajustar cómo se distribuyen e interactúan los biomas entre sí, permitiendo ajustar el tamaño general de los biomas y una curva para controlar las transiciones entre ellos.

- Puntos de interés: Permite añadir qué puntos de interés se quieren generar.
- Opciones de tipo booleano. Permite marca la opción que quiere que se cumpla. Si es una isla, si tiene puntos de interés, si el mapa se actualiza sin necesidad de generarlo de nuevo y si se genera vegetación.
- Botón de generar: Este botón al ser pulsado actualizará el mapa en la escena si es que se ha cambiado algún valor, borrando el anterior creado.

En las siguientes figuras (Figura 3.1, Figura 3.2, Figura 3.3, Figura 3.4) se muestran esquemas a seguir de los distintos componentes de la herramienta, en una ventana de Inspector de Unity.

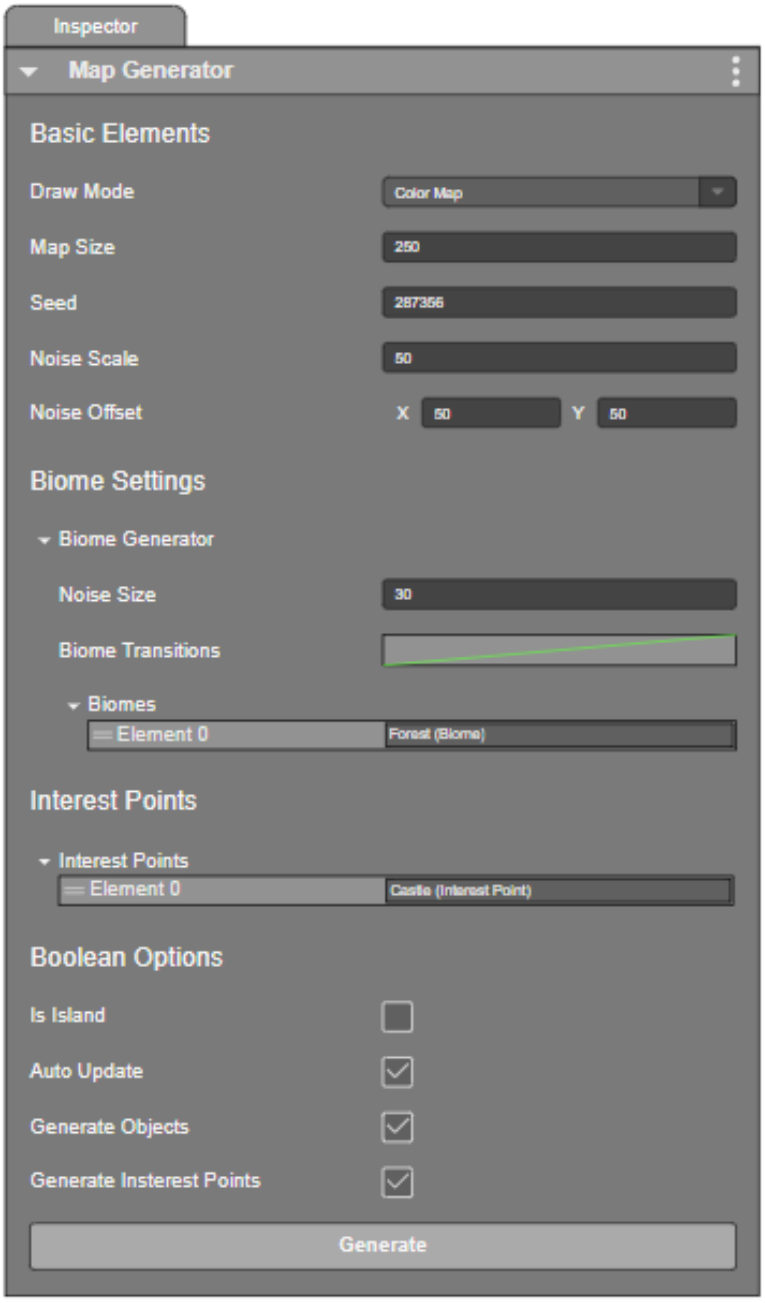


Figura 3.1: Primer diseño de la ventana de editor del componente MapGenerator

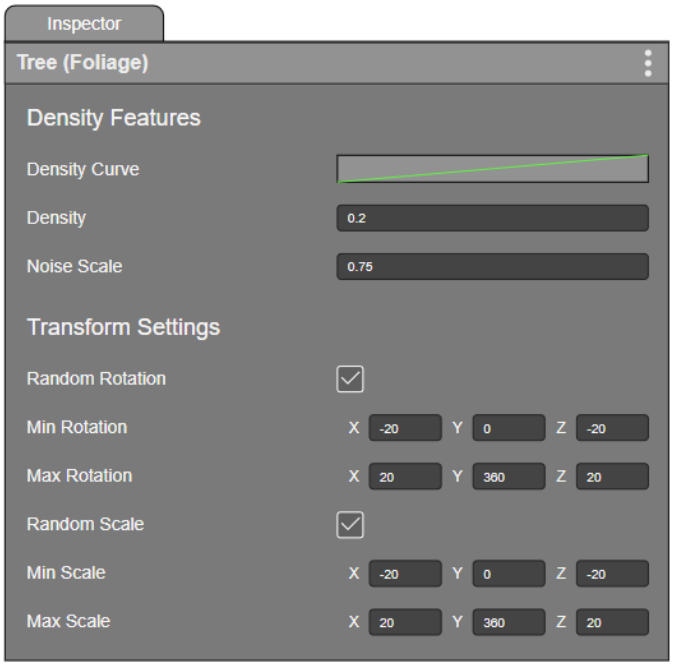


Figura 3.2: Atributos de los objetos tipo vegetación

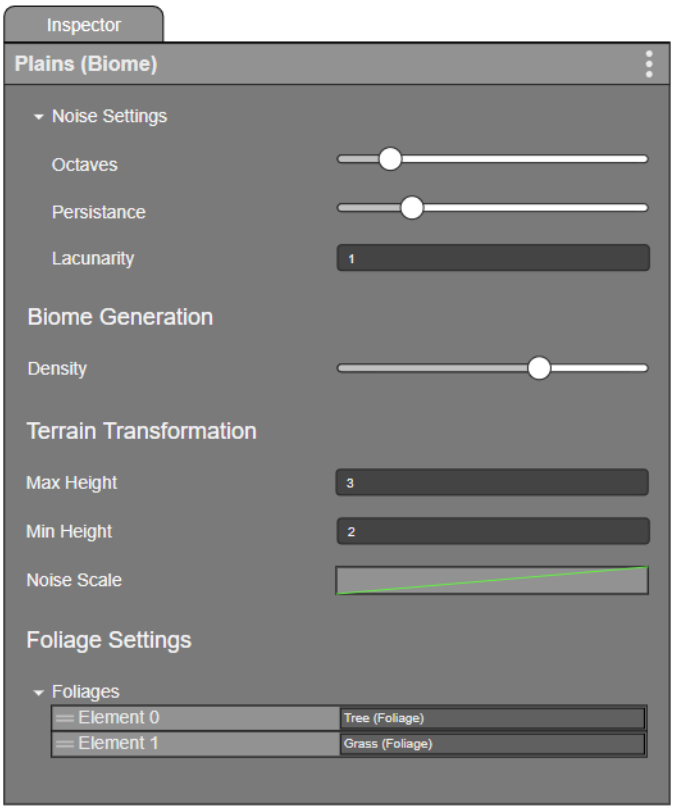


Figura 3.3: Atributos de los objetos biomas

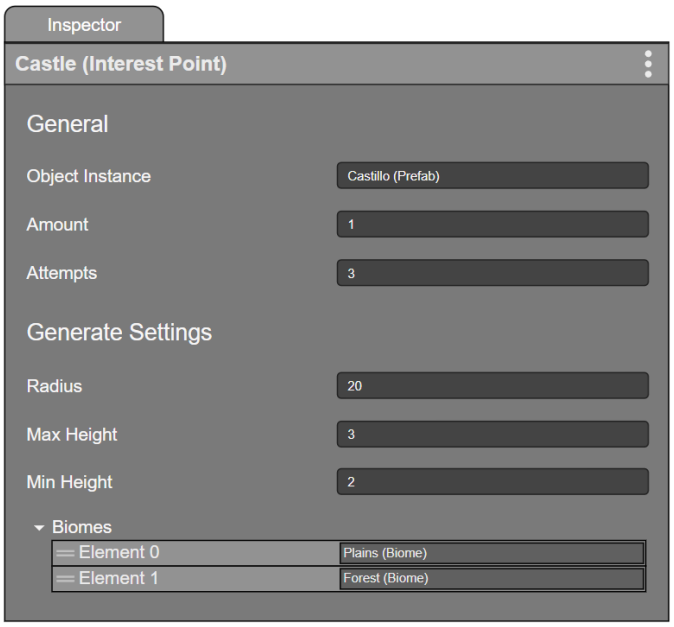


Figura 3.4: Atributos de los objetos de puntos de interés

Capítulo 4

Metodología

En este capítulo se habla del modelo de proceso y planificación general llevado a cabo para la realización del proyecto, las diferentes iteraciones e ideas que han ido surgiendo a lo largo de este y por último, se mencionan las herramientas utilizadas para aplicar la metodología planteada.

4.1. Modelo de proceso y planificación general

Para el desarrollo del proyecto hemos usado como metodología ágil de desarrollo **Scrum**, ya que esta proporciona una estructura flexible y colaborativa para el desarrollo. Además, aporta los siguientes beneficios:

- Fomenta la colaboración y el trabajo en equipo, lo que puede mejora la creatividad y la innovación.
- Permite adaptarnos rápidamente a los cambios en las necesidades.
- Proporciona un enfoque estructurado para la gestión del proyecto.
- Fomenta la transparencia y la comunicación abierta dentro del equipo.

Las metodologías ágiles son procesos que permiten adaptar la forma de trabajo a las condiciones del proyecto, consiguiendo flexibilidad e inmediatez en la respuesta para amoldar el proyecto y su desarrollo a las circunstancias específicas del entorno.

Aún así, para amoldarnos al periodo académico, es decir de los meses de septiembre hasta mayo con vacaciones de Navidad y Semana Santa, hemos estructurado el desarrollo del proyecto en tres etapas o hitos. Cada uno ha tenido una duración de aproximadamente 3 meses.

Al seguir un modelo Scrum utilizamos 'sprints'. Los 'sprints'son periodos relativamente cortos de tiempo destinados a que el equipo complete una serie de objetivos o tareas del proyecto. En nuestro caso cada sprint duraba 2 semanas, ya que esa era la frecuencia de las reuniones con nuestro director.

Durante cada hito se han completado las tareas necesarias par completar cada sprint. En cada una de las reuniones de estos 'sprints'se supervisaba el trabajo llevado a cabo y se encomendaban nuevas tareas a cada miembro del grupo. Al final de cada hito se hacía una presentación general entre todos los alumnos que trabajan en el grupo de investigación en la que teníamos que explicar nuestro proyecto y enseñar una demo si era posible.

- **Hito 1**, con una duración desde inicio de curso en septiembre hasta el 18 de diciembre. El objetivo de este hito era principalmente el desarrollo de la idea, que ha ido variando bastante a lo largo del tiempo hasta que se ha consolidado finalmente. Tuvimos que recortar partes, ya que el alcance era demasiado ambicioso. También lo dedicamos a realizar la investigación pertinente para saber qué herramientas y algoritmos utilizar durante el desarrollo, para formarnos completamente en el tema de generación procedimental de contenidos, para ver el funcionamiento de herramientas parecidas a nuestra idea de proyecto y cómo diferenciarnos de estas. También se realizó una pequeña maqueta en Photoshop para mostrar la organización y parámetros de la herramienta. Hacia el final de este hito se empezó a documentar todo el proceso de investigación en estas páginas.
- **Hito 2**, con una duración desde Navidad hasta el 20 de marzo. Este hito estuvo fundamentalmente dedicado al desarrollo de la herramienta en Unity. También arreglamos los errores que iban surgiendo y añadimos más funcionalidades que se fueron proponiendo. Para este hito también preparamos una demo completamente funcional de la herramienta probando a generar distintos escenarios.
- **Hito 3**, con una duración desde Semana santa hasta el 30 de mayo. Para este hito el proyecto estaba prácticamente estaba acabado. Así que se destinó a arreglar errores, optimizar la herramienta y asegurar su correcto funcionamiento. Se han creado también los cuestionarios y pruebas de evaluación que han sido entregados a distintos usuarios, para poder comprobar la utilidad del proyecto.

Todas las tareas de los *sprints* se pueden ver en nuestros 'issues'de Github del proyecto. Cada uno tenía una prioridad asignada y un coste que equivalía a una serie de horas. Se intento repartirlas de forma equitativa entre todos los miembros del equipo usando estimaciones.

4.2. Iteraciones y generación de ideas

Durante el proceso de desarrollo la propuesta esencial se mantuvo constante e inalterable. Se recortó contenido por su complejidad y se fueron

añadiendo nuevas ideas.

Primera Propuesta
La propuesta inicial era crear un proyecto de generación procedimental de mundos incluyendo biomas, ecosistemas y evolución inteligente de seres vivos en ese entorno. Sin embargo, en una fase muy temprana de primer hito, tras varias semanas de investigación y discutirlo con nuestro director, este proyecto al nivel de personalización que buscábamos era demasiado ambicioso. Teníamos que elegir si dedicar nuestro proyecto a la evolución de seres vivos en ecosistemas o llevar a cabo la generación procedimental de terrenos. Tras ver que el tema de generación procedimental aportaba más información y ejemplos, apostamos por esto ultimo y especializarnos en ello.

Segunda Propuesta
Esta propuesta se formuló durante el periodo del segundo hito. Continuó la línea de la primera propuesta pero se fueron añadiendo algunas ideas como consecuencia de habernos centrado en la generación de terrenos únicamente. Algunas de estas ideas fueron:

- **Ríos**. Durante la mitad del hito 2 surgió la idea de incorporar la generación de ríos dentro de la herramienta. Para ello se iba a utilizar un algoritmo llamado 'PerlinWorms'junto con A* para encontrar el mejor camino por el que circularía el río. Este proceso era muy costoso y los resultados no eran impresionantes, así que se desechó la idea. Después se intentaron implementar como un 'bioma'más, pero dependiente de la altura. Los resultados obtenidos fueron mejores. Sólo estaba implementado para la versión 2D del mapa generado, por lo que los resultados únicamente podían observarse ahí. Sin embargo, a la hora de implementar su representación gráfica en 3D, debido a su complejidad se terminó por descartar la idea.

- **Puntos de interés**. Al final del hito 2, nuestro director sugirió colocar ciertos puntos que podrían resultar de utilidad en un juego, como por ejemplo torres de piedra en un mundo medieval. Para implementar su creación de forma pseudoaleatoria dependiendo de un rango, se utilizó el algoritmo de *PoissonDisc*. De esta forma se crean los objetos designados (completamente configurables) en su rango específico y en una cantidad muy concreta por el mapa. Esta idea no se desechó y de hecho continuó en el proyecto.

- **Decoración**. Se sugirieron varias ideas de decoración del escenario preparado para la demo. Estas fueron: animales o personajes moviéndose por el escenario y una simulación de viento para los elementos de vegetación (hierba, árboles, etc.). Este paquete ya incluía animaciones y movimiento de los animales. Esto tiene una función simplemente decorativa para la demo. Sin embargo, la simulación de viento es de

cosecha propia y se incorporó al proyecto como un método más de personalización del mundo. Un detalle que le aporta vitalidad al mundo. Esta implementación se llevó a cabo mediante un 'shader'que se puede aplicar a los objetos que conforman la vegetación para dar esa sensación de viento. También se mencionó la posibilidad de incorporar un personaje que actuara de jugador que se fuera moviendo por el escenario. Se acabó incorporando e incluso se usó como objetivo del Endless Terrain.

Tercera Propuesta

La tercera y ultima propuesta se introdujo en el ultimo hito. En la cual simplemente se recorto el contenido que se muestra a continuación y se desarrolló una nueva funcionalidad mas útil para nuestra herramienta.

- **Ríos**. La funcionalidad que permitiría la creación de ríos en los terrenos resultó ser mas complicada de lo esperado, invirtiendo una cantidad mayor de tiempo del planeado inicialmente.
  Es verdad que se llegó a avanzar considerablemente en la generación de los ríos, por lo menos en cuanto a lógica (no tanto en cuanto a geometría), pero al final se decidió prescindir de esta funcionalidad tanto por temas de tiempo como por hallar casos no previstos en la planificación que llevaron a aumentar el tiempo de desarrollo de esta funcionalidad.

- **Posicionamiento por biomas del usuario**. También surgió una idea relacionada con una funcionalidad que es común en este tipo de videojuegos cuyo mundo es generado procedimentalmente. Esta funcionalidad se basa en la posición en la que debería aparecer el usuario cuando se genera un terreno. Si nos fijamos en videojuegos del estilo como Minecraft o Rust, el jugador aparece siempre en unos biomas específicos que no suponen un gran peligro (en Minecraft los bosques, llanuras, etc.) y en Rust las playas.
  Por esta razón, decidimos añadir una funcionalidad que permita a los usuarios, decidir en qué bioma específico quiere posicionar al jugador al generar un terreno nuevo.

- **Decoración**.

4.3. Herramientas Utilizadas

Unity

Durante el desarrollo de Armazón para Unity, hemos trabajado con la versión **2022.3.1f1** de Unity (véase en:Figura 4.1) como entorno principal con configuración URP (Universal Render Pipeline). Esta versión específica nos

proporcionó las herramientas y funcionalidades necesarias para llevar a cabo el proceso de creación de la herramienta. La escogimos principalmente por ser una de las últimas versiones LTS del mercado en el momento del desarrollo y así estar lo más actualizados posible. Usamos URP debido a la libertad artística que queremos proporcionar con la herramienta de creación de mundos. De esta forma aumentamos la personalización y permitimos el uso de más tipos de shaders.

Este motor tiene integrado como entorno de desarrollo Visual Studio. Utilizamos la versión 2022 en este caso. Al trabajar con este entorno de desarrollo en conjunto a Unity fue necesario usar el lenguaje de programación C#. Esto facilitó la implementación de la lógica y la interactividad necesarias para la herramienta.

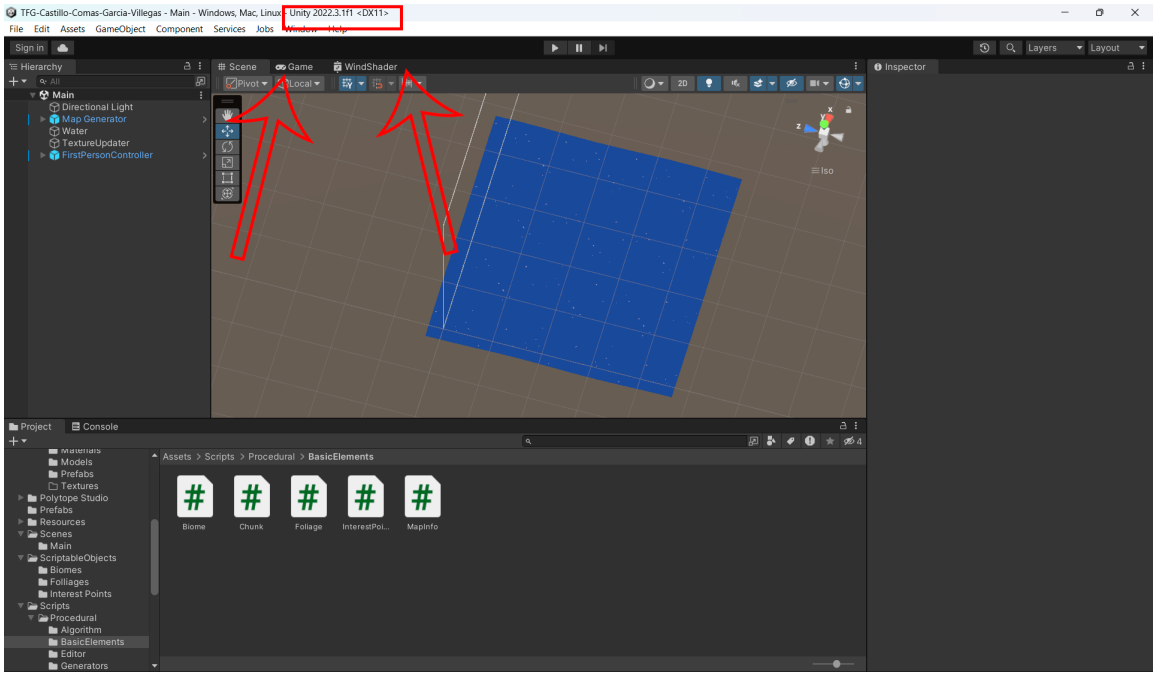


Figura 4.1: Versión de Unity

4.3.1. Github

GitHub (Microsoft, 2008). Es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. En ella tenemos un repositorio que contiene el proyecto de Unity y ahí se actualizan todos los cambios que hemos ido haciendo a lo largo del desarrollo.

Para no tener que usar la consola bash de git, hemos utilizado Github Desktop como interfaz gráfica, que además dispone de integración con GitHub.

De esta manera, nos resulta más sencillo realizar las tareas más básicas como crear o clonar repositorios, conectarlos con GitHub y subir o bajar archivos con nuestros cambios en el proyecto.

Hemos optado por aprovechar las funcionalidades y capacidades ofrecidas por GitHub para gestionar tanto el Product Backlog como el Sprint Backlog. En el Product Backlog, utilizamos la función de 'issues'de GitHub para crear y almacenar de manera organizada todas las tareas, funcionalidades y mejoras pendientes para el desarrollo. Cada 'issues'representa un elemento del Product Backlog y nos permite detallar la descripción, asignar etiquetas, definir prioridades y asignar responsabilidades de manera eficiente.

4.3.2. Slack

Slack (Slack Technologies, 2013). Es una herramienta de comunicación en equipo. Dispone de canales de texto organizadas por temas. Utilizada para facilitar las comunicaciones internas y coordinarnos con el director del trabajo. Utilizamos esta herramienta para programar reuniones, discutir temas relevantes y mantener una comunicación fluida y ágil entre los miembros del equipo.

4.3.3. Google Drive

Drive (Google, 2012). Se trata de un servicio de alojamiento de archivos compartido entre los integrantes del equipo. Lo hemos utilizado para almacenar documentos. Entre ellos las actas de las reuniones e información y enlaces relevantes fruto de la investigación previa hecha sobre la generación procedimental. Para hacer estos documentos usamos las propias herramientas de Google Docs. También usamos Google Slides para crear las presentaciones utilizadas durante los hitos.

4.3.4. Google Meet

Nuestras reuniones diarias, conocidas como Daily Scrum o Reuniones Diarias, fueron diseñadas para mantenerse dentro de un límite de tiempo de 15 minutos. Durante estas sesiones, el equipo se reunía para compartir de manera concisa y efectiva las actividades en curso y los logros alcanzados. Cada integrante tenía la oportunidad de exponer brevemente lo que estaban llevando a cabo y comunicar los avances realizados desde la última reunión. Este formato ágil y focalizado nos permitió mantenernos alineados, identificar posibles obstáculos y coordinar esfuerzos de manera efectiva sin prolongar excesivamente el tiempo de la reunión. Si bien las interacciones cotidianas y las reuniones internas se llevaban a cabo en Discord, las sesiones específicas

con nuestro tutor se realizaban en una sala designada de Google Meet. Esta elección se basó en la conveniencia y en las funcionalidades particulares que ofrecía Google Meet para ese tipo de encuentros más enfocados y de carácter más formal.

4.3.5. Overleaf

Overleaf (WriteLaTeX Limited, 2012) es un editor de LaTeX online. LaTeX es un sistema de composición de textos, orientado a la creación de documentos. Ha sido utilizado para la redacción de esta memoria de nuestro trabajo, tratando de conseguir más uniformidad en dicho documento.

Capítulo 5

Desarrollo

A lo largo de este capítulo se abordará el análisis, diseño y la implementación de la herramienta, la cual ha sido nombrada como **Diseño y desarrollo de un armazón para la generación procedimental de terreno y vegetación en Unity**.

5.1. Análisis y Diseño

5.1.1. Análisis

En esta sección se enumeran las historias de usuario propuestas para construir el proyecto. Es necesario definir estas historias al menos brevemente para poder estructurar el proyecto y repartir el trabajo entre los distintos miembros. Esto facilita el proceso de diseño que a su vez también facilita más tarde la implementación.

- Añadir el punto de partida. Existe un proyecto previo para Unity muy primitivo realizado por el miembro Javier Comas, el cual sirve de punto de partida para la herramienta.
- Configurar el tamaño de los bloques del prototipo cúbico. Es decir, los terrenos que se construyen a partir de cubos, tendrán un tamaño de bloques o de los cubos configurable.
- Añadir la opción al prototipo original de crear mallas mas suaves y lisas en vez de trabajar con un terreno cúbico.
- Renderizar las mallas con múltiples texturas y colores dependiendo de la altura, también con funcionalidad para difuminar los colores entre capas para que las transiciones queden mas suaves.
- Uso de LODs en las mallas del terreno generado, los LODS (Level Of

Detail) son una técnica de optimización que reduce la complejidad geométrica de los modelos 3D en función de su distancia desde la cámara.

- Creación de distintos biomas.
- Generación de objetos y vegetación.
- Los chunks del terreno deben solo renderizarse si están en el rango del jugador. Esto debe hacerse por temas de optimización en terrenos grandes.
- Implementar puntos de interés que se puedan repartir por el mapa.
- Movimiento de los objetos de tipo vegetación simulando el viento.
- Poder posicionar al objeto "jugador.[en] un bioma especifico al generar un terreno
- Configuración del Editor, es decir, la interfaz en la que aparecen los parámetros para que sean modificables por el usuario de la forma mas cómoda posible.

5.1.2. Diseño

A continuación se describirá detalladamente el diseño software correspondiente a cada historia de usuario de las mencionadas anteriormente. Mediante el uso de imágenes y diagramas explicativos trataremos de documentarlo de manera sencilla.

5.1.3. Añadir punto de partida

El punto de partida ha sido un proyecto realizado previamente por uno de los autores del trabajo, *Javier Comas De Frutos*. Este mapa sólo se genera con una forma cúbica en su forma 3D. Además permite generar distintos mapas en 2D a modo de previsualización, tanto del ruido como del resultado final.

Como extra, en el punto de partida también contiene un shader de agua, el cual simula el movimiento del mar y sus mareas.

Este punto de partida se va modificando a lo largo del desarrollo a la hora de llevar a cabo el resto de especificaciones.

5.1.4. Configurar el tamaño de los bloques

El objetivo de configurar el tamaño de los bloques, es hacer que todos los bloques del mapa cúbico tengan las misma diferencia de altura de un bloque con respecto a sus adyacentes, es decir, que un bloque siempre tenga una

cierta altura con respecto a otro. Por otro lado, con respecto al tamaño, el usuario debe poder determinar qué tamaño de bloque quiere para el mapa, además de la altura pero siempre manteniendo una relación constante para mantener la coherencia del mapa, es decir, la relación entre los tamaños de los bloques y las alturas que mantienen estos con respecto a otros.

Para ello, se debe dar en el editor la opción al usuario de poder modificar el tamaño de los bloques.

5.1.5. Crear mallas suavizadas para el terreno

Se partió de un proyecto en el que el terreno que se generaba estaba compuesto por cubos. Para ampliar este herramienta se crean terrenos más definidos y suavizados que simulan un terreno más realista.

Para poder llevar a cabo esta funcionalidad se debe igualar cada uno de los vértices creados a su altura correspondiente en el mapa de alturas. Primero se dará la opción en el editor de elegir que tipo de terreno se quiere generar, si se elige el suavizado, en el componente encargado de la generación de mallas se hará esto.

5.1.6. Renderizar las mallas con múltiples texturas y colores dependiendo de la altura

En el desarrollo de nuestra herramienta de generación de terrenos procedimental, nos enfrentamos al desafío de crear entornos visuales dinámicos y atractivos que además no sean complejos de utilizar y personalizar.

Para abordar este reto, implementamos un sistema que permite renderizar las mallas del terreno con múltiples texturas y colores, dependiendo de la altura de cada punto del terreno.

Para realizar esta funcionalidad, el usuario debería ser capaz de dividir el terreno por capas de altura. Cada capa de altura debería tener una textura asignada y un color con el cual tintar la textura.

También es necesaria una funcionalidad para poder combinar las texturas entre capas mediante un difuminado y que sea parametrizable.

Para realizar toda esta funcionalidad, haremos uso de un shader que renderize texturas por capas de altura y un script aparte para poder asignar esas texturas al shader y todos los valores que tengan que ver con esta funcionalidad.

5.1.7. LODs

Como se ha mencionado en el Capítulo 2 la función de los LODs es reducir el número de vértices y de polígonos de la malla del terreno. Como el terreno cúbico esta construido con formas geométricas básicas y con un número de vértices reducido no será necesario aplicar niveles de detalle. En

cambio, para los terrenos lisos sí que resulta útil, ya que posee una geometría bastante más compleja.

Para aplicar lo niveles de detalle usaremos el componente con esta misma función del que ya dispone Unity, llamado LOD Group. Este componente admite que se le asocien a los distintos niveles varias mallas. En nuestro caso cada nivel tendrá una malla. De otra forma el rendimiento caería drásticamente al generar montones de mallas para el terreno. La totalidad de vértices para mapas de tamaño muy superior sería inmanejable. Las mallas del terreno son generadas por el generador de mapas, por lo que a la hora de generar esas mallas generaremos 3 tipos, una para cada nivel de detalle. Por tanto, se debe modificar el generador de mallas para que de acuerdo a un nivel de detalle u otro construya el mismo terreno pero con distinto número de vértices.

A la hora de generar las diferentes mallas de cada sección del terreno, el generador de mallas recibe un parámetro que indica cuánto se debe simplificar la malla resultante, y a partir de dicho valor, ser capaces de crear la malla con más o menos vértices. Esto lo conseguimos reduciendo el número de valores del mapa de alturas general que utilizaremos para cada malla según lo que hayamos definido anteriormente, siendo dicho parámetro cada cuántos valores del mapa de alturas general vamos a utilizar para crear un vértice.

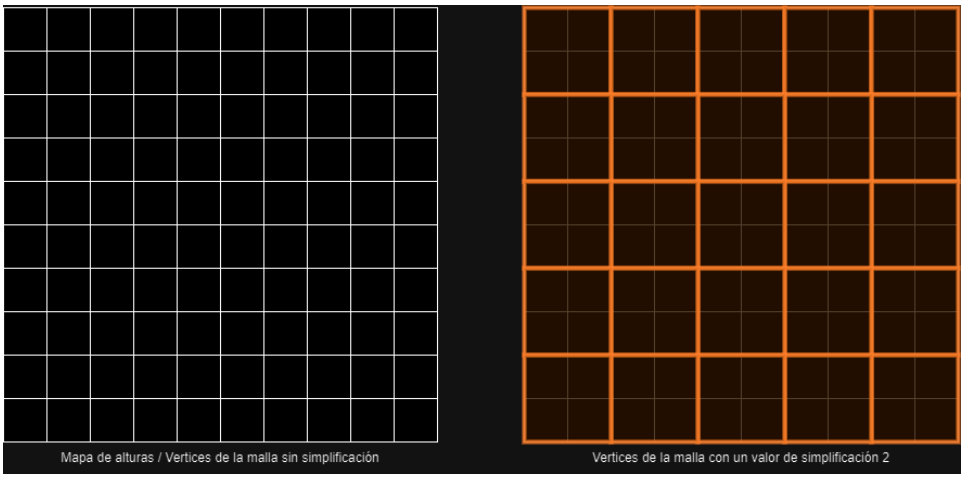


Figura 5.1: Visualización de cómo se eligen los valores para la malla

5.1.8. Biomas

Los biomas, como se menciona anteriormente, sirven para ofrecer mayor variedad de texturas y formas del terreno al mapa generado.

Tenemos que ser capaces de ofrecer al usuario una gran variedad de op-

ciones de personalización en este aspecto, ya que son los biomas los que al fin y al cabo decidirán como se genera el terreno del mundo. Tenemos que dar la opción de personalizar la formación del terreno de cada bioma, por lo cual damos acceso a parámetros que se utilizarán a la hora de crear el mapa de ruido utilizado para el terreno, además de darles la capacidad de elegir cómo quiere que se procesen esos valores de ruido con una curva de animación de Unity, utilizando el eje x como el valor de cada casilla del mapa de ruido y en el eje y, el resultado, el valor que realmente se usará en la generación del terreno. Unido a esto también deberíamos dar la posibilidad de ajustar las alturas máximas y mínimas posibles para cada bioma, siendo estos los valores mínimos y máximos de la curva mencionada anteriormente.

A la hora de colocar los biomas en un mundo, utilizaremos otro mapa de ruido de Perlin, pero al contrario que en los biomas, no damos la posibilidad al usuario de ajustar todos los valores, ya que consideramos que sería demasiada información a procesar, y sólo damos acceso al tamaño del mapa de ruido, así ajustando el tamaño general de cada bioma en el mundo. Para poder generar un terreno medianamente natural, tenemos que ser capaces de generar terrenos utilizando valores de varios biomas, y para decidir que puntos van a tener mas influencia un bioma u otro, necesitamos otra curva de animación de Unity que podrá ajustar el usuario, para gestionar la transición de terreno de un bioma a otro.

5.1.9. Generación de objetos y vegetación

La generación de objetos, como hemos comentado anteriormente, sirve para poder darle vida al mundo que estamos creando. Remarcando lo anterior, nosotros entendemos por vegetación un objeto que queremos instanciar por el mundo o por parte de él. Queremos que el usuario pueda instanciar los objetos que necesite por el mundo con múltiples configuraciones, permitiendo así que el usuario pueda definir como quiere sus objetos dentro del mundo sin la necesidad que los coloque a mano y sin embargo, los pueda reposicionar. La configuración que nosotros queremos dar al usuario a parte de las típicas configuraciones de rotación, escala, altura aleatorias, es la posibilidad de que el objeto se adapte al entorno, es decir, que se le aplique un factor de rotación y posicionamiento en función de la posición en el mapa. Por ejemplo, a un objeto en una colina empinada, podría tener que adaptarse a la colina y que el objeto esté inclinado según la colina, más la posible rotación aleatoria que tenga.

Un factor también muy importante es la distancia de separación con otros objetos, ya que no es lo mismo poner una roca que igual alrededor no quiere objetos, que césped que no requerirá distancia de separación con otros objetos. A su vez otro parámetro bastante relevante sería cuanto del objeto queremos que se hunda en el suelo (o que lo esté sobrevolando). Esto nos permitiría tener un mayor control sobre el posicionamiento del objeto en el

suelo.

Además otro punto importante a tratar es donde queremos que se generen, por ello cada bioma ha de tener los objetos que puede generar en su terreno y con una determinada densidad que indique la cantidad de objetos que puede llegar a generarse de un tipo. A su vez, otro factor importante es determinar a qué altura es más probable que se genere un objeto ya que esto da más juego en cuanto a la generación, e infinitas configuraciones a la hora de generarlos. Lo último, pero no por ello menos importante, es el factor de aleatoriedad. Hemos decidido usar Perlin Noise o Ruido de Perlin, al igual que en los terrenos, ya que este nos puede aportar un nivel de detalle y realismo. Por ello otro factor a tratar sería una pequeña configuración de ese Perlin Noise con una escala para el ruido para poder aumentar la variabilidad a la hora de generar los objetos.

De esta forma el objeto que queremos crear tendría esta estructura con los siguientes parámetros que determinarían la configuración del objeto que vamos a ir instanciando por el mapa generado.

- Prefab Properties:
  - Prefab: Objeto que se quiere instanciar
  - Require Distance Separation / No Require Space Separation : por si el objeto requiere distancia de separación
  - Unit Space Separation: Unidades de Unity de separación que requiere ese objeto con respecto a otros
- Density Features:
  - Density Curve: Curva que permite establecer de que parte a que parte hay mas probabilidades de que se genere el objeto
  - Density: Densidad del objeto que queremos instanciar
  - Noise Scale: escala del ruido de Perlin
- Transform Properties:
  - Rotation:
    - Random Rotation: Si el objeto va a tener rotación aleatoria
    - Min Rotation: Rotación mínima del objeto
    - Max Rotation: Rotación máxima del objeto
    - Rotation: Si el objeto no va a tener rotación aleatoria, quÉ rotación básica va a tener el objeto
  - Scale:
    - Random Scale: Si el objeto va a tener escala aleatoria
    - Min Scale: Escala mínima del objeto

    - Max Scale: Escala máxima del objeto
    - Scale: Si el objeto no va a tener escala aleatoria, quÉ escala básica va a tener el objeto

- Advanced Settings:

    - Enviroment Rotation: Si queremos que el objeto se adapte al terreno, es decir, posicionarse correctamente con respecto a este
    - Subsistence in the ground: Cantidad del objeto que se puede hundir en el suelo

5.1.10. Endless Terrain

Esta herramienta puede llevarse a cabo de distintas formas. Estas son:

- **Generar y eliminar los bloques**. El algoritmo construye de cero la malla de los bloques basándose en los parámetros del generador procedimental de terrenos que se encuentran en el rango de visibilidad. Los bloques que hayan sido generados, una vez que salen de ese rango son eliminados. En el caso de que el jugador vuelva a una ubicación en la que ya haya estado, se vuelve a generar el terreno.
- **Activar y desactivar**. En este caso, al detectar si alguno de los bloques en los que se divide el mapa está en el rango de visión del jugador los hace visibles (activa el objeto). En cambio si se encuentran fuera del rango, simplemente los hace invisibles (desactiva el objeto).

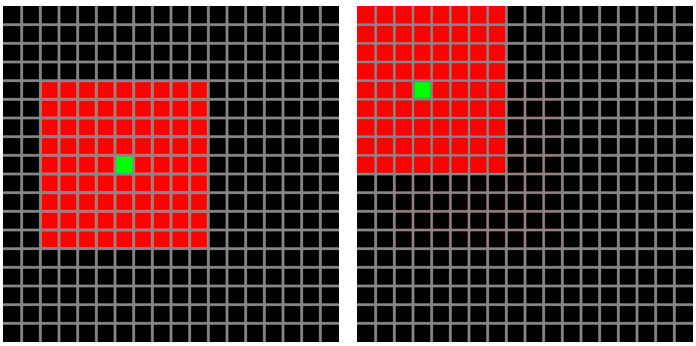
En nuestro caso optamos por activar los chunks cercanos al jugador y desactivar los más lejanos. Se parte de un mapa generado previamente que seguirá la configuración establecida a través de los parámetros del MapGenerator. Por lo que no existe un mapa infinito, si no uno simplemente de gran tamaño. El jugador podrá ver los límites del mapa si se acerca a ellos tal y como pasa en el juego Minecraft, donde se utiliza esta herramienta de forma similar, sin embargo, consta de un mapa mucho más grande.

Debido a que depende del movimiento del jugador, esta herramienta no funciona en el Editor, si no que únicamente funciona en ejecución.

En un ejemplo de un mapa de tamaño de 900x900 con un tamaño de chunk de 50 y una máxima distancia visible de 200, se vería de este modo: (Figura 5.2)

5.1.11. Simulación de viento

La simulación del viento es un aditivo que se hizo en el proyecto en una fase bastante avanzada del desarrollo. Es una función decorativa para



(a) (b)

Figura 5.2: Ejemplo de Endless Terrain

dotar de vida al mundo que se genera. Esta simulación se aplica a todos los elementos que conforman el vegetación. En vez de hacer una simulación física real, lo cual sería bastante costoso en cuanto a rendimiento, se utiliza un shader. Como los shaders permiten modificar la posición de los vértices de una textura, se puede aplicar a todos los elementos de tipo vegetación sin ningún problema, ya que todos ellos poseen o poseerán una textura y un material al que se le puede aplicar cualquier shader.

Para conseguir este movimiento se siguen tres pasos:

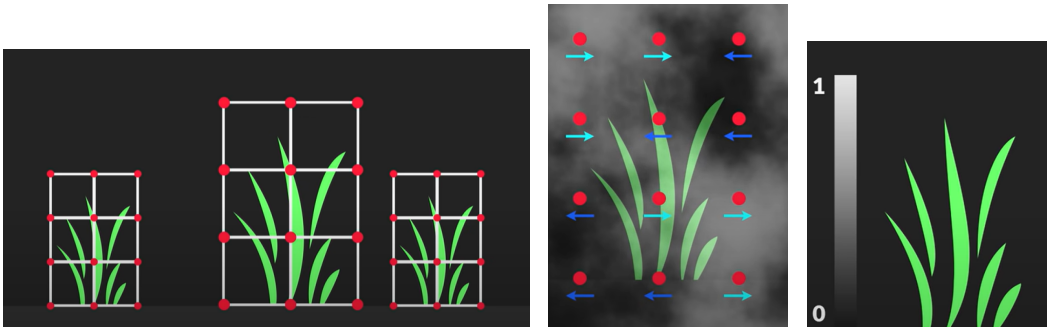
1. Obtenemos la posición global de los vértices del objeto afectado. Usamos las posiciones en X e Y para crear UVs e inclinarlas en el tiempo.

2. Se genera algo de ruido en global también basado en las UVs. Basándose en ese ruido se puede empujar o tirar de la textura en el eje de las x. Por ejemplo, para valores bajos se empuja y para los valores altos se tira.

3. Para evitar un movimiento raro del objeto (se movería desde abajo y daría un efecto extraño) hay que asegurarse que la base el objeto se mantiene fija. Para ello se crea un gradiente del 0 al 1, siendo el 0 la parte más baja y el 1 la parte más alta. El 0 indica movimiento nulo, y el 1 movimiento total. Para ello se usa el eje de las y de las UVs.

5.1.12. Puntos de interés

Para colocar los puntos de interés se hace de forma semi-procedimental. Se colocan de forma aleatoria y uniforme dentro de un o unos biomas seleccionados y en un rango de alturas, evitando zonas despobladas o superpobladas. Esto se consigue separando todos los puntos con una distancia mínima, todos los puntos estarán separados entre ellos por un radio configurable.



(a) Paso 1 (b) Paso 2 (c) Paso 3

Figura 5.3: Pasos para crear el shader Brackeys grass sway in Unity - shader graph (https://www.youtube.com/watch?v=L_Bzcw9tqTc)

Estos se generan sólo si su opción correspondiente está marcada en el generador de mapas. Se crean en la parte del mapa 3D.

Todos los objetos se construyen en base a un objeto modelo que tiene una serie de parámetros que afectan en su generación. Cada objeto debe tener su propio modelo. Es importante recalcar que estos objetos no se solapan con el resto de objetos del mapa previamente creados ya sean puntos de interés o vegetación.

Se puede ver un ejemplo de pseudocódigo en el Apéndice F.

5.1.13. Posicionamiento del jugador en un bioma específico

Al igual que en algunos juegos en el que se generan terrenos procedurales, el personaje que controla el jugador suele aparecer en un bioma específico. Este bioma suele ser llanuras con césped, que representan menos peligro para el jugador desde un inicio.
En esta herramienta también se da soporte para este tipo de funcionalidad. Se ha construido un sistema para que al generarse un terreno, automáticamente el jugador se teletransporte a una posición aleatoria del terreno especificando un bioma en concreto. Por ejemplo, un desarrollador podría hacer que su personaje, al generar un nuevo mundo, este aparezca en el bioma de desierto, creado por él anteriormente.

5.1.14. Configuración del Editor

Como queremos que sea lo más cómodo posible para el usuario, hemos decidido modificar la previsualización de diversas partes de nuestros scripts para que sea todo mas visual.

En general hemos decidido agrupar por parámetros que podemos decir que están dentro de una determinada utilidad, con un título para que sean mas fáciles de encontrar. Por ejemplo, en la vegetación las características que modifican la posición, escala y rotación del objeto.

En la Figura 5.4, encontrará una versión simplificada del diagrama de clases.

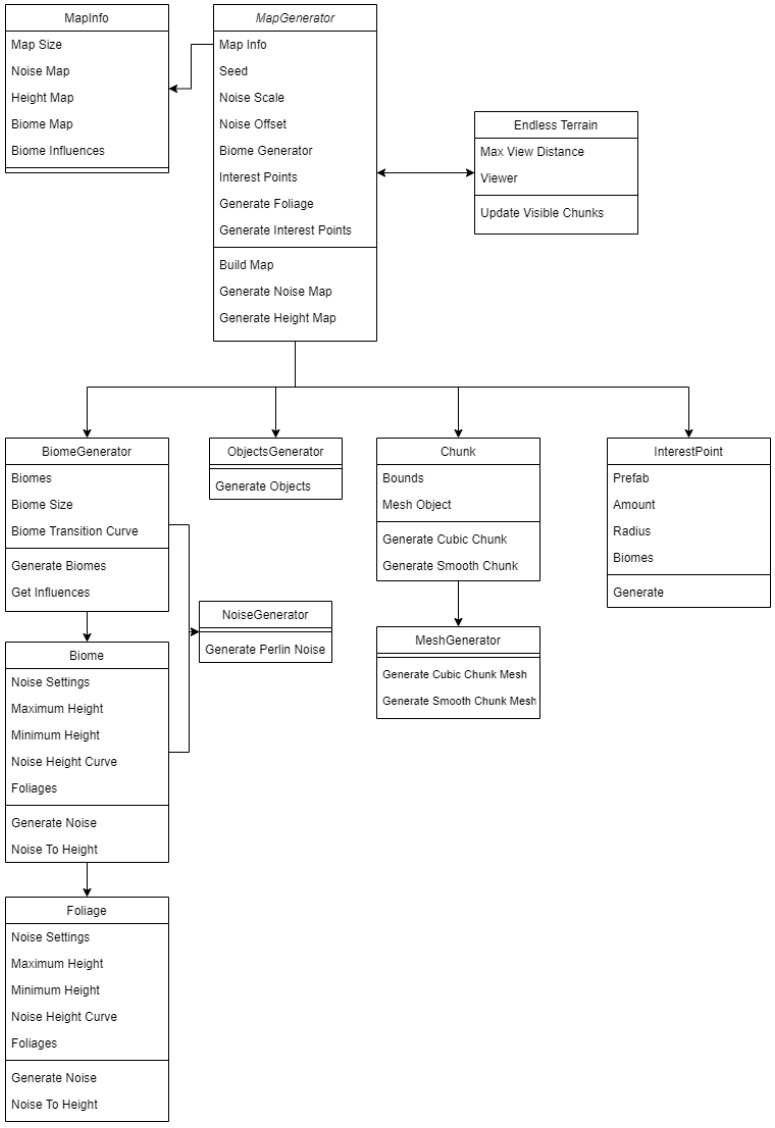


Figura 5.4: Diagrama de clases simplificado

5.2. Implementación

En esta sección se exponen los detalles técnicos de las tareas o historias de usuario implementadas para crear la herramienta.

5.2.1. Añadir punto de partida

El punto de partida cuenta con un gameObject principal con un script MapGenerator, el cual dispone de parámetros para la configuración de:

- Tamaño del mapa (int mapSize).
- Factor de escala del ruido generado (float NoiseScale). Un valor mayor producirá un ruido con detalles más precisos.
- El número de octavas utilizadas en el algoritmo de ruido (int Octaves). Cada octava es una capa de ruido que se suma al resultado final. A medida que se agregan más octavas, el ruido generado se vuelve más detallado y complejo.
- Amplitud de cada octava (float Persistance). Un valor más bajo reducirá el efecto de las octavas posteriores de las octavas posteriores.
- Un multiplicador que determina qué tan rápido aumenta la frecuencia para cada octava sucesiva en una función de ruido de Perlin (float Lacunarity).
- Número aleatorio utilizado para generar el ruido (int Seed).
- La posición inicial del ruido generado y su desplazamiento (Vector2 Offset).
- Capas del terreno que se pueden generar (TerrainType[]regions).
- Objetos que se pueden generar a lo largo del mapa (ObjectInMap[]objects).
- Generar un suavizado en el ruido generado de tal forma que el mapa tenga forma de isla (bool useFallOff).
- Cuando se realiza un cambio desde el editor, se actualiza automáticamente el mapa (bool autoUpdate).
- Cuando se inicializa este componente auto, se regenera el terreno (bool autoRegenerate).

Con todos estos parámetros, se puede generar el mapa cúbico 3D o también da la oportunidad de poder generar el mapa 2D a modo de previsualización.

MapGenerator implementa la funcionalidad de:

La generación de los mapa de alturas tal y como ha sido descrito en el Capítulo 2, aunque solo se genera un único mapa de alturas por mapa, el cual se usa posteriormente para construir la malla. Para la generación de la malla existe un sistema de chunks. Existe un modo isla que permite aplicarle una capa de alturas al mapa de alturas dando así la sensación de que el mapa generado es una isla. Para la creación de esta capa "Island" se uso de la fórmula que se puede ver en Figura 5.5 .

Esta fórmula se utiliza para modelar como la intensidad o la cantidad de

$$f(x) = \frac{x^a}{x^a + (b - bx)^a}$$

Figura 5.5: fallOff Formula "Decaimiento radiactivo"

una magnitud disminuye a medida que nos alejamos de la fuente.

En cuanto a la generación de mallas, la malla cúbica, divide el terreno en dos mallas: edges y suelo, para evitar la solapamiento de vértices debido a la limitación de Unity de vértices por malla. Para la generación de vértices se parte de que cada punto del mapa de alturas crea 4 vértices, para ello se toman los índices de acceso a la matriz como referencia y se le aplica el tamaño de bloque, que, como hemos dicho antes, era estático, más X por defecto de desplazamiento provocado por el tamaño del mapa. A medida que se crean los vértices se van creando los triángulos de la malla, cada uno de los cuales se compone de tres vértices identificados por el índice de la lista en la que estamos almacenando los vértices creados. Estos se crean en el sentido horario del reloj para que luego la iluminación refleje correctamente sobre la malla. Sin embargo, la altura de cada bloque no es la misma para cada bloque porque no hay redondeo ni ninguna otra técnica para fijar siempre que la distancia de altura entre cada bloque sea la misma.

Para la generación de las paredes es exactamente lo mismo, pero solo se generan vértices en el caso de que la casilla en la que queremos generar su anexa es decir actualX+-1 o actualY+-1 sea de altura inferior a la actual.

La generación de terreno dispone de un struct TerrainType en el cual todos los puntos de altura que estén a una determinada altura se ponen del color de ese Terrain. El editor tiene un array de este struct y hay que ponerlos de menor a mayor, es decir, de 0 a 1, para que luego en la creación de la textura del mapa (textura que se crea simplemente con estos colores), se ponga cada píxel del color de terreno que le corresponde. La creación de la textura es muy simple, se crea una textura y se crea un array de colores de tamaño de mapa y dependiendo del mapa de alturas se aplica el color del terreno pertinente.

Como se ha comentado anteriormente, la generación de objetos es bastan-

te primitiva, existe una clase a modo “vegetación” que te deja establecer las características de densidad en la capa de terreno que se genera y el noisescale para la generación “aleatoria de los objetos”.

5.2.2. Configurar el tamaño de los bloques

Para configurar el tamaño de los bloques, tenemos que dividirlo en dos partes. Por un lado, la altura de cada bloque que para que fuese simétrica hemos cogido el mapa de ruido y lo hemos redondeado con dos decimales, una vez redondeado el mapa de altura con dos decimales y obtenido el mapa de altura con la influencia de los biomas, redondeamos a un decimal para obtener una escala de altura que vaya de 0 a 1 y multiplicamos por 10 para obtener una altura mas constante en términos de tamaño, y esto multiplicado por el tamaño de bloque se adapte al tamaño de bloques del usuario. De esta forma la altura máxima capaz de alcanzar nuestro mapa y por tanto un bloque es de 10 veces el tamaño de bloque.

Por otro lado el tamaño de bloque en sí desde el punto de vista de la representación física, en las constructoras de la mallas cubicas se multiplica la posición de los vértices por el tamaño de los bloques consiguiendo así el tamaño de bloques deseado, a su vez como aumenta la distancia entre vértices también hay que desplazar el chunk por tamaño de bloque para compensar este incremento, en cuanto al chunk, nos referimos al objeto padre del chunk que contienen la malla y objetos de esa porción del terreno.

5.2.3. Crear mallas suavizadas para el terreno

Para la generación de la malla suavizada hemos creado en Mesh Generator un método que se encarga de la conversión de los puntos de altura a vértices para la creación de la malla. Para ello al igual que para la creación de la malla cubica generamos una malla por chunk para no sobrepasar los limites de vértices por malla. De esta forma accedemos al mapa de altura con las posiciones correspondientes de cada chunk y por punto de altura creamos un vértice con posición de acceso a la matriz junto con sus correspondientes incrementos de posición dependientes del tamaño de mapa para que se coloquen bien, y de altura el punto de altura calculado anteriormente en el mapa de alturas. A medida que vamos creando los vértices vamos creando los triángulos de la malla los cuales cada uno se componen de tres vértices identificados por el índice de la lista en la que estamos almacenando los vértices creados, estos se crean en el sentido horario del reloj para que luego la iluminación refleje correctamente sobre la malla. Por ultimo establecemos todos esos vértices, triángulos, UVs , todos estos en forma de array en el componente tipo Mesh. Además, recalculamos las normales de los vértices para que la iluminación y sombreado en la malla se calculen correctamente.

5.2.4. Renderizar las mallas con múltiples texturas y colores dependiendo de la altura

Para lograr la funcionalidad de renderización por altura con texturas y colores parametrizados en Unity, se ha tenido que hacer uso tanto de codigo en lenguaje C# para gestionar tanto la parametrización y la configuración básica. El tronco de la funcionalidad reside en su mayoría en un Shader el cual se encarga del procesamiento y renderización de la información especificada. Dicho shader es especifico del pipeline URP (Universal Render Pipeline). URP es un sistema de renderizado optimizado y adaptable en Unity para crear gráficos de calidad en una variedad de dispositivos y plataformas.

Aquí se detalla cómo implementamos esta integración:

5.2.4.1. Lógica en C#

En el lado de la lógica en C#, un script se encarga de administrar los parámetros de texturas, colores y alturas para cada capa de altura definida en el terreno, y del proceso de traspasar los datos elegidos por el usuario a variables útiles dentro del código del shader escrito en lenguaje HLSL.

Todos los parámetros que definen las capas se especifican creando y personalizando un ScriptableObject de Unity. Cada capa de altura contiene los siguientes parámetros :

- Texture Texture: Textura que se va a utilizar en esta sección del terreno, esta textura debe ser tileable y accesible y modificable, por lo que el sprite debe importarse con una configuración especifica, asignando el parámetro ReadOnly a false. Cabe resaltar que no es obligatorio asignar una textura, la herramienta entonces tendrá en cuenta el color elegido.
- Color Tint: La herramienta también te da la opción de tintar la capa, combinando el color seleccionado con la textura. También es una opción no asignar un color, en este caso no se modificará el color de la textura.
- float TintStrength: Se constituye como un valor decimal entre 0 y 1. Si el valor es cercano al 0, el color tendrá menos efecto sobre la textura mientras que si es cercano al 1, la textura se vera mas afectada.
- float StartHeight: Este parámetro se encarga de definir la altura desde la que se empieza a renderizar esta capa. Constituye un valor decimal entre 0 y 1, siendo 0 el punto mas bajo del terreno y 1 el punto mas alto del terreno.
- float BlendStrength: También constituye un valor entre 0 y 1. Define el suavizado de la textura de esta capa con el resto de capas. Si el valor es cercano al 0, el cambio entre las texturas de las capas es inmediato,

sin ningún tipo de suavizado, mientras que si es cercano al 1, las demás capas se verán afectadas por esta de forma mas notable.

- float TextureScale : Define la escala con la que se renderizará la textura de esta capa.

5.2.4.2. Shader URP programado en HLSL

La funcionalidad básica consiste en el cálculo de textura por altura: Implementamos lógica para calcular, para cada punto en el terreno, qué textura y color deben utilizarse en función de su altura, e influencias de cada capa en cada punto. Esto implica iterar sobre todas las capas definidas y determinar el color en el punto actual, considerando el suavizado entre capas.

Las secciones mas importantes de esta funcionalidad son el "Vertex Shader" y el "Fragment Shader", a continuación se explican brevemente porque son necesarios y para que se utilizan específicamente en este shader.

Vertex Shader (vert): Su función principal es transformar los vértices de los objetos en la escena desde su posición en el espacio de objetos al espacio de pantalla, donde serán proyectados y finalmente renderizados en la pantalla.

Este shader comienza definiendo la estructura de entrada (appdata) que contiene la posición del vértice, las coordenadas de textura y la normal del vértice. Luego, define la estructura de salida (v2f) que contiene la posición del vértice en el espacio de pantalla, las coordenadas de textura, la posición del vértice en el mundo y la normal del mundo. En la función vert, se transforma la posición del vértice al espacio de clip y se calculan las coordenadas de textura y la posición del vértice en el mundo. También se transforma la normal del mundo al espacio del objeto.

Fragment Shader (frag)
Es el responsable de determinar el color final de cada pixel que se renderizará en la pantalla, así como de aplicar una variedad de efectos visuales para lograr la apariencia visual deseada. Se ejecuta por cada píxel renderizado en pantalla.

Comienza calculando el valor normalizado de la altura del fragmento en relación con la altura mínima y máxima del terreno. Seguido del cálculo de una variable "blendAxes" teniendo en cuenta las normales globales del fragmento para aplicar el efecto "triplanar"al shader.

Triplanar Mapping: Es una técnica de mapeo de texturas utilizada para aplicar texturas a objetos en una escena tridimensional de manera que se eviten las distorsiones visuales en superficies con diferentes orientaciones. En lugar de depender exclusivamente de las coordenadas de textura UV de la geometría, el triplanar mapping proyecta la textura desde tres ejes orto-

gonales (X, Y, Z), combinando las proyecciones para cada eje para obtener un mapeo uniforme y sin distorsiones.

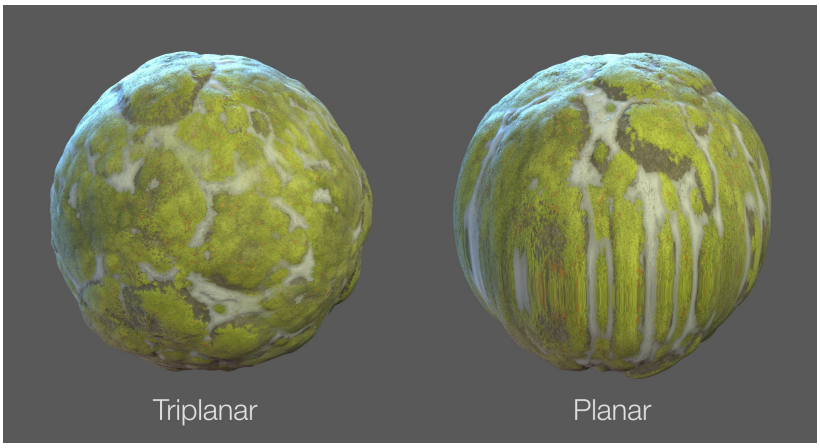


Figura 5.6: Efecto visual del "Triplanar mapping"

Por cada fragmento, se iteran todas las capas, en cada iteración se calcula:

- La influencia que tiene dicha capa sobre este punto.
- El color base de la capa actual teniendo en cuenta el color asignado y la fuerza del mismo.
- El color que tiene el fragmento según la textura de la capa, en este paso se hacen cálculos mas complejos ya que aquí es donde se aplica la técnica triplanar.
- El color que tendría este punto según la capa actual, todos estos colores se van añadiendo al color final del fragmento.

Con todos estos cálculos realizados, se obtiene el color final para este punto en la pantalla. Este proceso se realiza igual por cada fragmento hasta renderizar la pantalla entera.

5.2.5. LODs

A la hora de crear un chunk con distintos niveles de LOD, lo conseguimos saltándonos parte de los valores de altura del mundo, de tal forma que conseguimos reducir el número de vértices, mientras mantenemos la forma general del terreno, que es justo lo que necesitamos. Para saber cuantos valores del mapa nos tenemos que saltar, necesitamos una lista de divisores del tamaño del chunk, ya que si utilizamos cualquier número, podría ocurrir que un chunk no se genere correctamente generando huecos entre ellos, ya que al intentar saltarse un número de valores, se salga de los límites del chunk sin

haber generado sus bordes. Para ello tenemos un método que nos devuelve todos los divisores de un número en un array ordenados de menor a mayor, y una vez lo tenemos, según el nivel de detalle que queremos, elegimos uno de esos niveles y ese será el número de valores que nos vamos a saltar para tener en cuenta para la malla. Llamaremos a éste valor el valor de simplificación.

Ahora sí para crearlo, conseguimos el mapa de alturas general del generador de mapas, que contiene la altura de cada posición del mapa en su totalidad. También tenemos los límites del chunk que tendremos en cuenta del mapa de alturas. Definimos la posición inicial de los vértices como la mitad del tamaño del mapa total, tanto en los ejes horizontal como vertical. Para generar una malla cualquiera necesitamos dos arrays diferentes: uno que contiene todos los vértices de la malla, y otro con los índices de los vértices de tres en tres y ordenados para formar los triángulos. Ahora recorremos los valores del mapa de alturas que se encuentran dentro de los límites del chunk actual en dos bucles for. En cada valor añadimos un nuevo valor al array de vértices y añadimos dos nuevos triángulos si es posible, formando el cuadrado que tenga como una de las esquinas el vértice nuevo. Cada vértice no es más que una posición en un espacio tridimensional cuyas posiciones x, z vendrán definidas por su posición en el mapa de alturas mientras que su valor en y será el valor de la casilla del mapa de alturas. Al meter los LODs en el algoritmo, simplemente vamos incrementando los índices de los bucles for por el valor de simplificación, que al ser divisor del tamaño total del chunk, generará el chunk del tamaño que corresponde, evitando así agujeros entre chunks por tener tamaños erróneos.

El componente *LOD Group*, por defecto aplica funcionalidad LOD, Los LODS (Level Of Detail) en Unity son una técnica de optimización que reduce la complejidad geométrica de los modelos 3D en función de su distancia desde la cámara, mejorando el rendimiento del juego al mostrar versiones simplificadas de los objetos cuando están lejos.
Los diferentes niveles de un objeto con LOD, representan diferentes mallas del mismo objeto, cada una con menos vértices que el anterior.
Este componente tiene 3 niveles de LOD y cuando se aleja lo suficiente ya deja de mostrarse por completo. Cada nivel tiene un array, inicialmente vacío de renderers, los cuales deben tener asignada una malla con distinto número de triángulos de acuerdo a su nivel. En nuestro caso no será necesario que el array de renderers tenga más de tamaño 1, solo generaremos una malla por nivel.

Las mallas de un terreno procedimental no pueden estar guardadas en el proyecto como un prefab. Estas mallas deben construirse nuevas cada vez que se genera un terreno. Por todo esto el componente LOD Group de Unity debe añadirse por código.

En apartados anteriores se ha explicado cómo se generan las mallas (por chunks). Es por eso que el componente LOD se añadirá al objeto *floor* de

los diferentes chunks, ya que es el que contiene la malla. Como queremos 3 niveles distintos creamos un array de LOD de tamaño 3. Para cada LOD crearemos un gameObject hijo de floor (mediante un bucle for que recorre este array) al que se le aplicarán todas las características de los LODs. Debe hacerse de esta forma, en vez de aplicarlo directamente al objeto floor. Esto se debe a que Unity necesita un ciclo para añadir un componente, y en cambio todo este proceso se hace en un bucle que se ejecuta en el mismo ciclo. Esto provoca que se guarde la misma malla para todos los niveles. Lo mismo ocurre con el MeshCollider, al ser un collider que se adapta a la forma de la malla, y esta va a ir variando, se debe hacer de la misma forma. A continuación se genera una malla para ese objeto hijo de floor con un nivel de detalle cuyo valor irá aumentando en cada vuelta del bucle para hacer mallas menos complejas. Una vez hecho esto, se crea un array de tamaño 1 de MeshRenderer al que se añade el MeshRenderer del hijo y a continuación se construye el LOD en el array de LODs con ese array de MeshRenderer como parámetro. Una vez el bucle haya acabado se guarda el array de LOD al componente de LOD Group. De esta forma el objeto padre tomará las mallas de los objetos hijos gracias al componente LOD Group.

5.2.6. Biomas

A la hora de generar los biomas, primero decidimos donde van a estar localizados cada uno. Para ello, utilizamos una clase llamada BiomeGenerator, que genera y guarda la información necesaria de la posición de los biomas. Contiene un método que recibe el tamaño del mapa que se desea generar. Inicializa un array bidimensional de bytes que contendrá el índice del bioma que corresponda en cada posición del mapa. A su vez, creamos otro array bidimensional del mismo tamaño que inicializamos con un mapa 2D de ruido de perlin, con valores entre 0 y 1. Cada bioma tiene un atributo de densidad, que se irán añadiendo a otro array de manera sumativa, de forma que le ultimo elemento del array será la suma de todos los valores de densidad de los distintos biomas. Ahora, para cada valor del ruido de Perlin, se multiplica por la suma de las densidades, para que acto seguido podamos comparar los valores del mapa de Perlin multiplicado con los del array de densidades, para averiguar entre que dos valores cae, y así saber entre que dos biomas se encuentra nuestro punto del mapa. Si el valor del mapa de Perlin modificado es menor que el primer valor del array, consideramos que se encuentra en el primer bioma, sin influencia de ningún otro.

La influencia de un bioma se representa como un diccionario en C#, las claves siendo referencias a un bioma, y los valores como un número en punto flotante entre 0 y 1 para cada bioma. La suma de todos los valores siempre debe ser 1.

Para calcular la influencia de cada posición, simplemente vemos la distancia de nuestro valor del mapa de ruido modificado a los dos valores dentro

de los que cae, cuanto mas cerca esté de uno e los valores, más influencia tendrá el bioma asignado a ese valor sobre la posición. Esa diferencia la normalizamos de tal forma que los dos valores son entre 0 y 1, además de que su suma sea 1. Antes de asignar el valor, lo pasamos por una curva asignada por el usuario para que pueda elegir cómo manejar las transiciones. Ahora sí, una vez pasados el valor por la curva, los volvemos a normalizar y ahora si se asignan como influencia de su respectivo bioma.

Ahora, ya en el MapGenerator, generamos el mapa de ruido de Perlin de cada bioma del tamaño del mapa entero utilizando los atributos de cada bioma, y con ello y el mapa de influencias del BiomeGenerator, podemos generar el mapa entero, utilizando los valores de cada bioma de acuerdo con su influencia en cada zona, de tal forma que también transición entre ellos.

5.2.7. Generación de objetos y vegetación

Para la construcción del sistema de vegetación lo primero que hemos hecho es un nuevo scripteableObject en el cual establecemos toda la configuración comentada anteriormente. Para los parámetros del scriptableObject de vegetación hemos usado booleanos, vector2, animationcurve, floats e ints.

Para la generación de objetos cada bioma posee una lista de objetos que se pueden generar en su tipo de terreno de esta forma podemos conseguir que cada bioma por así decirlo sea un tipo de terreno exclusivo en el que se pueden generar un tipo de objetos.

Hemos creado una clase estática llamada Objects Generator, la cual contiene los métodos encargados para la generación de objetos. En primer lugar, dependiendo del modo que se este ejecutando encontramos dos grandes métodos, uno para cuando generamos un mapa normal con el editor y otro para el Endless Terrain. Tenemos que distinguir ya que el Endless Terrain solo genera los chunks cargados por el jugador y por lo tanto no podemos generar todos los objetos de golpe, al contrario que el generado de mapa normal que genera y carga todos los chunks del golpe. El método de generación de .Endless Terrain"genera los objetos para un chunk, pero el modo de generar los objetos es igual en los dos métodos.

Para generar los objetos vamos recorriendo como hicimos anteriormente el mapa de alturas y lo primero que hacemos es ver si podemos generar un objeto en esa posición. Para ello tenemos una lista de objetos y comprobamos si esa posición esta libre. Si esta libre comprobamos a que bioma pertenece ese punto para saber que posibles objetos podemos generar en ese punto. Una vez sabemos que es posible generar ese objeto en ese punto, obtenemos una lista de los objetos de ese bioma ordenados por orden de densidad y calculamos un noiseValue multiplicando la X y Y de posición del mapa de alturas por el noiseScale. Este valor lo comparamos con un random hecho entre 0 y el valor obtenido con las densidades del objeto (densidad del objeto

y la densidad del objeto teniendo en cuenta el punto de altura). Si el noiseValue es menor que la densidad podemos colocar el objeto. Luego calculamos la posición relativa en el mapa del objeto. Si estamos en modo cartoon y el usuario ha activado el enviromentRotation lanzamos un rayo para abajo del objeto para obtener la rotación y posición del objeto con respecto al suelo de tal forma que se adapte el objeto a la posición del suelo. Una cosa a tener en cuenta es que a la hora de generar los objetos, como estamos generándolos nada mas después de generar la malla del mapa incluyendo colliders y demás, tenemos que esperar un frame para poder lanzar este raycast y obtener la posición y rotación con respecto al suelo, ya que desde código cuando aun gameobject le añades un collider tarda 1 frame hasta que se llega a establecer al completo. Independientemente para el cálculo de la rotación aleatoria hacemos un random entre los dos valores establecidos por el usuario y, en el caso de que necesitemos adaptar el objeto al terreno con la técnica explicada anteriormente, usamos un qlerp para respetar la rotación random establecida anteriormente y la posición y rotación con el suelo. Para la escala random es el mismo proceso que con la rotación. Por último, se llama a un método para que si el objeto necesita distancia de separación con otros ocupe la distancia de separación necesaria para ese objeto, de tal forma que no se puedan generar mas objetos alrededor de este.

Para explicar la generación de objetos vamos a suponer que estamos en el modo "generar todos los objetos del mapa", en el caso del chunk sería igual pero solo recorriendo las posiciones de ese chunk determinado.

5.2.8. Endless Terrain

Es necesario crear un gameObject en la escena que actúe como el jugador para obtener su posición. Al principio comenzó siendo un objeto vacío y luego se sustituyó por un personaje en primera persona que se mueve por el escenario.

Tras esto se crea un componente llamado Endless Terrain (Script) que será el encargado de la funcionalidad y que se asocia al MapGenerator. Este script tendrá como variables públicas en el editor la posición del jugador, para facilitar su asociación y la distancia máxima de visibilidad para hacerla configurable.

Lo primero que hace es generar un mapa internamente a través del MapGenerator y calcular los chunks visibles dividiendo la distancia máxima de visibilidad entre el tamaño de los chunks. De esta forma trabajamos con coordenadas, por lo que se debe pasar la posición del jugador también a coordenadas, dividiendo los dos componentes de su posición (x e y) entre el tamaño de los chunks.

Mediante dos for anidados recorremos todos las coordenadas que se en-

cuentran en una distancia visible y que se encuentran dentro de los límites del mapa generado previamente. Hemos de tener en cuenta que este mapa está a nivel interno y que ninguna malla o gameObject ha sido generado en la escena. Por ello debemos llevar un registro de los chunks. Si un chunk se encuentra en una coordenada válida para renderizarse y no ha sido registrado previamente se crea su malla y su gameObject llamando a la constructora de Chunk. En cambio, si se encuentra registrado y cumple las condiciones, si está desactivado (porque el jugador pasó por ese lugar, pero luego se alejó) se activa. El método que contiene toda esta funcionalidad se llama cada frame y en cada frame se desactivan los chunks activados para actualizarlos de manera más eficiente. También se actualiza constantemente la posición del jugador, ya que es la variable de la que depende todo el funcionamiento de este algoritmo.

Para saber la visibilidad de un chunk, en el script Chunk hay dos funciones: una booleana que devuelve el estado del gameObject (activo o no) y otra que permite, a través de un parámetro, cambiar su visibilidad. Además este script también tendrá una función Update que en función de la posición del jugador y la distancia máxima de visibilidad oculta o no ese chunk. Como todos los chunks tienen este componente (script) se puede tratar de forma individual al visibilidad de cada uno.

Al ubicarse prácticamente toda la funcionalidad en el Update del componente, se entiende que sólo funcionará si se ejecuta Unity. Por ello y para evitar problemas, se bloquea la generación de un mapa con el botón Generate del MapGenerator si este componente está activo.

5.2.9. Puntos de interés

Para llevar a cabo la generación de los puntos y su colocación en el mapa se utiliza el algoritmo de *PoissonDisc*. Este algoritmo consiste en posicionar objetos de forma pseudoaleatoria por un mapa colocándolos con una distancia mínima entre cada punto, al cual se llama radio, ya que es una distancia alrededor del objeto. Se van añadiendo los puntos de forma aleatoria y se comprueba la distancia entre ellos. Si un nuevo punto añadido está en el radio de alguno de los puntos ya colocados, este se descarta y se vuelve a intentar. Para construir esto a nivel interno, se crea una cuadricula o rejilla, en la que la diagonal de cada casilla representa el radio de distancia entre objetos. Por ello no importa en qué punto de una casilla esté el objeto, se garantiza su separación y significa que sólo hay espacio para un punto en cada casilla.
También se descarta un punto si su altura en el mapa correspondiendo a esa posición en 2D que se le asigna en el algoritmo no se encuentra en el rango de alturas proporcionado por el usuario o si el bioma asociado a esa posición no está en la lista de biomas asociada a ese punto de interés en concreto.
Para implementar este algoritmo es necesario especificar a través de pará-

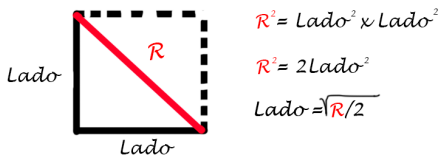


Figura 5.7: Pitágoras

metros el tamaño del mapa, el radio de dispersión, el número de instancias, la cantidad de comprobaciones, además de la altura mínima y máxima para su generación (permite crear el rango) y una lista de biomas. Tras esto se guardan estas variables y se calcula el tamaño de las casillas mediante Pitágoras (véase en Figura 5.7), ya que se conoce la diagonal de la casilla (radio de dispersión) y se coloca un elemento en el mapa. Siguiendo la lógica anterior se van colocando los puntos por el mapa, asegurando que la posición candidata para un nuevo punto está dentro del mapa, en el rango de alturas, en el bioma o biomas pertinentes y si está a la distancia adecuada de cada punto ya puesto. Los puntos colocados se registran en una lista. Si la lista ya tiene el número especificado para ese objeto concreto no se colocan más. Las comprobaciones representa el número máximo de veces que se repetirá este proceso si una posición no es válida.

Con esto obtenemos una lista de posiciones para los puntos de interés. Después se crea un gameObject con el nombre asociado al modelo de ese objeto (es decir, si ese Interest Point se llama Castillo, el nombre del gameObject creado es Castillo) dentro del map3D que actúa como padre de todos los interest points de ese mismo tipo. A continuación se instancian todos esos puntos en el mapa a la altura adecuada en esas posiciones y se registran en la lista de objetos ya existentes del mapa para que no se solapen con el resto.

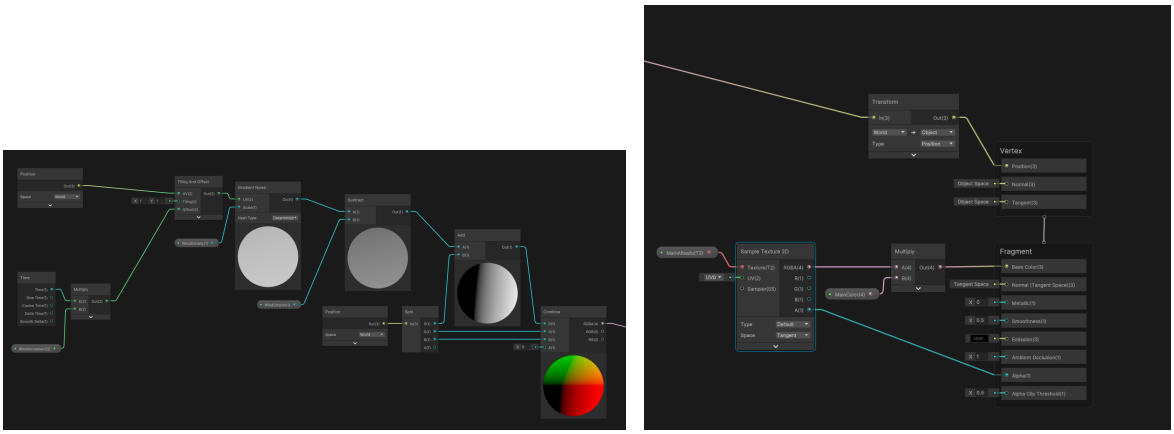
Los puntos de interés necesitan de un modelo del que partir para poder instanciarse. Este modelo es un scriptable object de Unity. Este objeto posee toda la información necesaria para la generación de ese punto, así como su prefab asociado (este prefab esta formado por la malla del objeto, su material y textura y en caso de necesitarlo, otros componentes), su cantidad (número de instancias en el mundo de tipo int), el radio de dispersión, el número de comprobaciones (int), el rango de alturas (ints) y un array de biomas (bime[]). Cada tipo de punto necesitará su propio scriptable object con su correspondiente configuración.

5.2.10. Simulación de viento

Como se ha mencionado previamente en el apartado de diseño se utiliza un *shader* para simular el viento en el mundo generado.

Este *shader* podría entrar en el tipo *Vertex Shader*. Esto se debe a que modifica las posiciones de los vértices del objeto al que se le asigna, deformándolo de tal forma que simula la agitación provocada por el viento. Se zarandea de un lado a otro basándose principalmente en tres parámetros: el movimiento del viento (Vector 2D), su fuerza (float) y su densidad (float). El movimiento del viento define los ejes donde se produce el movimiento (dirección) y hasta qué punto. Como el movimiento es horizontal no diagonal o vertical, solo se mueve en el eje de las X. Pero estas variables son públicas y se pueden modificar desde el editor si es que se quiere cambiar la dirección en la que sopla el viento. La densidad repercute en la brusquedad del movimiento. Y en este caso la fuerza equivaldría a el desplazamiento con respecto al centro del objeto que lo posee.
Este *shader* tiene otra parte dedicada al color (*Pixel Shader*). En esta parte se le asigna una textura y un color base con el que se mezclará. Para la hierba, por ejemplo, se mezcla la textura de hierba con el color verde. Y para los árboles cuya textura ya contiene los colores, basta con establecer el color base a blanco para que no afecte a los de la textura.

Para su programación se usó la herramienta integrada en Unity de *Shader Graph*, esta herramienta permite a los desarrolladores crear un *shader* de forma visual usando nodos sin tener que programar código. Además posee un sistema de feedback visual en el que se reflejan de manera instantánea los cambios y cómo estos afectan al resultado final. Puede observarse en: Figura 5.8



(a) Viento (b) Textura y Color

Figura 5.8: Shader de simulación de viento

Como se observa en la imagen, para simular el viento es necesario modificar el campo Position de Vertex. Para empezar con el paso 1 de la lista de pasos en el apartado de diseño necesitamos un nodo Position que nos apor-

tará esa posición global. El nodo Tiling and Offset permite inclinar las UVs. Para crear el efecto deseado, queremos que esto se efectúe en el tiempo, por ello necesitamos un nodo Time. Para poder controlar esa frecuencia tenemos la variable WindMovement, la cual se multiplica por el tiempo. Todo esto nos dará la inclinación.

El siguiente paso es crear el ruido, para ello creamos el nodo Gradient Noise. Su escala puede modificarse gracias al uso de la variable WindDensity. El nodo Subtract se añade para no obtener solo valores positivos, ya que de esta manera nada tiraría de la textura, solo se empujaría. Junto con la variable WindStrengh permite controlar la fuerza de ese movimiento. Para continuar añadimos un nodo Position para obtener las posiciones actuales de los vértices y el nodo Split para separar ese vector por componentes. Esto nos permite modificar sólo el eje X. Sumamos esto al cálculo anterior. Para finalizar usando el nodo Combine, se combina todo ese resultado final junto con el resto de componentes del vector de posición y se aplica a la posición del nodo Vertex.

Como se ve en la imagen b), para aplicar el color y la textura debemos mezclar un color en formato rgb con una textura 2D. Se ha creado una variable llamada MainAlbedo de tipo Texture 2D, a la cual se le asigna la textura que se desea aplicar al objeto en cuestión. Para poder mezclar la textura con un color debemos transformarlo a un Vector4 ya que es el formato en el que se trabajan los colores (rojo, azul, verde y el alpha, este último se trata a parte). Para ello hacemos uso del nodo Sample Texture 2D. Para poder mezclar ese resultado con un color (variable pública llamada MainColor que permite seleccionar el color deseado desde el editor) es necesario el uso del nodo Multiply.

5.2.11. Posicionamiento del jugador en un bioma especifico

La forma en la que se consigue esta funcionalidad es, una vez terminado de generar el terreno, se generan puntos aleatorios dispersos por el mapa, los cuales se procesan para saber a que casilla especifica pertenecen, y a su vez conocer la influencia de cada bioma que afecta a esta casilla.
Si la influencia del bioma seleccionado en una casilla es mayor que un número entre cero y uno especificado por el usuario, se convierte en una casilla valida y se posicionara el jugador en ella.

La forma en la que se especifica toda esta funcionalidad es mediante los siguientes parámetros modificables por el usuario:

- PlayerTransform: Constituye una referencia al personaje jugable dentro del juego. Este es el componente que se posicionará cada vez que se genere un nuevo terreno.
- Biome: Bioma en el que aparecerá el jugador cada vez que se genere un terreno procedimental.

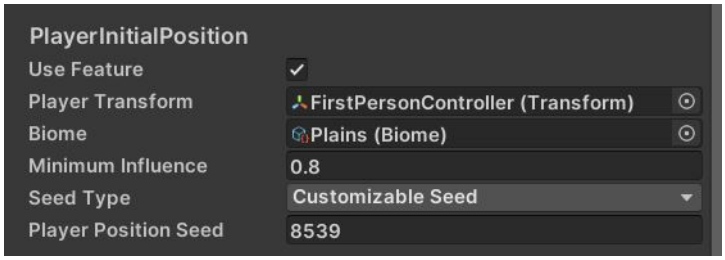


Figura 5.9: Vista del inspector

- MinimumInfluence: Este valor representa cuál es la influencia mínima del bioma seleccionado que debe tener una casilla para que sea apta para posicionar el personaje en ella.
  Por ejemplo, si se selecciona un bioma de desierto, y una "mínima influencia" de 0.7, significaría que el personaje se posicionará en una casilla que como mínimo tenga un 70 % de influencia del bioma de desierto.

- SeedType: Su funcionamiento tiene que ver con el uso de semillas para aleatorizar la posición del jugador. Este parámetro es un enumerador que tiene tres posibles valores:

  - GlobalSeed: Para un terreno generado, el jugador siempre aparecerá en la misma posición, esta suele ser la opción mas típica en este tipo de videojuegos (como por ejemplo -Minecraft).
  - RandomSeed: Genera una semilla aleatoria cada vez que se genera un nuevo terreno. Esto significa que aunque se genere el mismo terreno varias veces, el jugador puede aparecer en diferentes posiciones aleatoriamente.
  - CustomizableSeed: Con esta opción, se puede asignar una semilla a mano.

5.2.12. Configuración del Editor

Hemos usado el sistema de editor de Unity que nos permite modificar el "aspecto" de un script en el editor de Unity. Hemos utilizado la propiedad CustomEditor para poder realizar estos cambios. Para la creación de los Headers hemos usado un LabelField con el texto que queríamos de header y la letra en negrita con la propiedad EditorStyles.BoldLabel, por último para la recolocación de los parámetros hemos usado el PropertyField por orden según el orden que queríamos para los parámetros y el FindProperty para encontrar el parámetro dentro de la clase que estábamos customizando su editor.

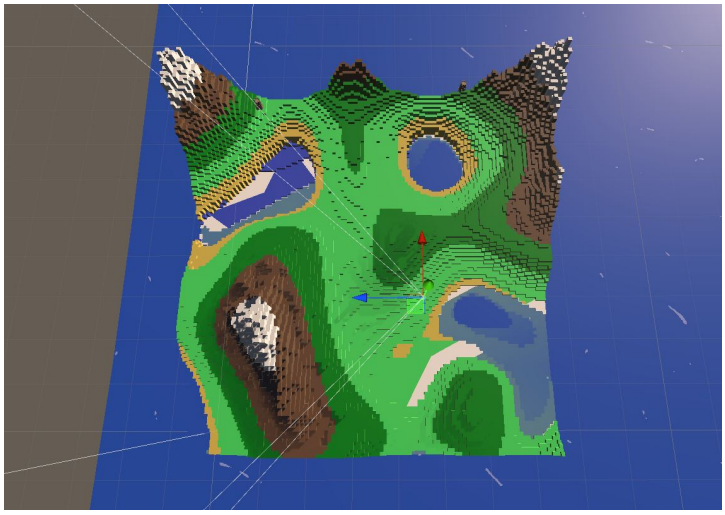


Figura 5.10: Configurado para que el jugador siempre se posicione en biomas de planicie al generar el terreno

Capítulo 6

Resultados

En este capítulo explicamos cómo se han desarrollado las pruebas y los resultados que estas nos han proporcionado, discutiéndolos brevemente.

6.1. Pruebas

Las pruebas están destinadas a desarrolladores independientes, es decir, que no trabajan en grandes estudios ni han llevado a cabo grandes proyectos. No se tiene en cuenta ni sexo, ni región de procedencia, pero el rango de edad está orientado a los 20-30 años.

El diseño de las pruebas está estructurado en la definición de unos objetivos que se validan con preguntas de investigación, las cuales se responden en el análisis de los resultados. La afirmación de estas preguntas sirve para validar el correcto funcionamiento de la herramienta.

Para ayudar a los usuarios a crear su mundo, se han añadido varios recursos, es decir objetos a modo de prueba, todos estos provenientes de la Asset Store de Unity.

Animales: https://assetstore.unity.com/packages/3d/characters/animals/5-animated-voxel-animals-145754

Vegetación: https://assetstore.unity.com/packages/3d/environments/landscapes/low-poly-simple-nature-pack-162153

A continuación se muestra el esquema del diseño de las pruebas.

6.1.1. Hipótesis y preguntas de investigación

En esta sección se enumeran los objetivos propuestos para la creación de las pruebas de evaluación de la herramienta y una explicación sobre estas.

6.1.1.1. Hipótesis

Comprobar que la herramienta es intuitiva para la creación de terrenos.

6.1.2. Hipótesis 1.1

El usuario comprende correctamente el uso y el comportamiento de los biomas.

6.1.3. Preguntas De Investigación

- ¿El usuario comprende cada uno de los parámetros del BiomeObject?
- ¿El usuario entiende el funcionamiento del Influence Range del Biome Generator?
- ¿El usuario entiende el funcionamiento de las curvas de transición del Biome Generator?

6.1.4. Hipótesis 1.2

El usuario comprende el funcionamiento y comportamiento de la vegetación.

6.1.5. Preguntas De Investigación

- ¿El usuario comprende cada uno de los parámetros del FoliageObject?
- ¿El usuario sabe cómo puede meter un objeto de vegetación en un bioma?

6.1.6. Hipótesis 1.3

El usuario comprende el funcionamiento de los shaders

6.1.7. Preguntas De Investigación

- ¿El usuario sabe cambiar la configuración del scripteableObject de shaders para que se dibuje todo a su gusto?
- ¿El usuario manipula las transiciones entre shaders?

6.1.8. Hipótesis 1.3

El usuario comprende el funcionamiento del endless Terrain

6.1.9. Preguntas De Investigación

- ¿El usuario activa el componente de Endless Terrain?
- ¿El usuario ejecuta el proyecto sin darle al botón Generate del Map Generator?

6.1.10. Duración y entorno de realización

La duración de la prueba es aproximadamente de 20 minutos. El tiempo final dependerá de la manera de desenvolverse de los probadores con la herramienta. Los primeros 5 minutos han servido para explicar al usuario el funcionamiento de herramienta. Se destinarán 10 minutos a observar al probador y ayudarle a manejar la herramienta en caso de que lo necesitara, mientras intenta crear un paisaje similar a uno que se le mostrará durante la explicación. Este es el tiempo más variable. Los últimos 5 minutos consistirán en una pequeña entrevista con el probador para obtener información sobre la herramienta y su experiencia con ella. Para más información sobre la entrevista véase el Apéndice E

Toda la prueba se realizará con un ordenador portátil con el entorno, hardware y software preparado antes de cada prueba. No se cuenta en estos 20 minutos el tiempo que se tarda en preparar de nuevo el ordenador tras cada prueba.

6.1.11. Descripción de las tareas del probador

El usuario escuchará una explicación ligera de la herramienta para que adquiera unos conocimientos mínimos a la hora de probarla. Acto seguido dispondrá de aproximadamente 10 minutos para que pruebe la herramienta en función del caso que se le asigne, siguiendo las instrucciones dadas. Cada uno deberá imitar un paisaje distinto dentro de esta selección preparada:

(a) Islas

(b) Montañas

(c) Volcán en la nieve

(d) Desierto

(a) Meseta

(b) Castillo

(c) Molino

(d) Faro

En caso de que el usuario tenga cualquier duda con respecto a cualquier parte de la herramienta puede preguntar a los investigadores. En caso de que la generación de algún mundo se le dificulte, recibirá ayuda de los investigadores, ya sea en forma de pista o ayuda directa. Una vez esté conforme con

sus resultados, este responderá a las preguntas de la entrevista final que le hará el investigador.

6.1.12. Instrucciones iniciales

El investigador selecciona una imagen a imitar. Escogiendo preferiblemente aquellos paisajes que no hayan sido utilizados anteriormente.
Se le enseñará al probador la función de todos los parámetros de la herramienta y a su vez se le darán indicaciones de ejemplo de estos parámetros. Además se pedirá al probador que incluya objetos en el mundo asignado, estos ya a elección del probador.
Tras esto se le deja total libertad al probador para que intente imitar el paisaje seleccionado.

Debido a la cantidad y complejidad de los parámetros, es normal que el probador no se acuerde de la funcionalidad de todos de manera que durante toda la prueba el probador puede preguntar para qué sirve cada parámetro y se le contestará al igual que en la explicación inicial, es decir, cuál es su funcionalidad junto a un ejemplo.

Cuando el probador manifieste estar conforme con su creación el investigador pasará a la fase de la entrevista. Anotará toda la información en el documento asignado para ello.

Tras esto hará una captura de pantalla al resultado de la prueba, la guardará y limpiará el proyecto de cara a iniciar otra prueba con otro usuario distinto.

6.1.13. Recogida de información

Durante toda la prueba se irán recogiendo anotaciones por parte del investigador sobre las acciones, preguntas y resultados que va obteniendo el probador. Esto se puede encontrar en el Apéndice E, donde aparecen las entrevistas a los probadores junto con los comentarios adicionales, donde se recogen todas estas anotaciones.

6.2. Resultados

Para medir los resultados a nuestras preguntas de investigación hemos elaborado una tabla para facilitar las comparaciones de resultados obtenidas de cada probador. Esta tabla está construida en base a las preguntas de investigación propuestas en el diseño de las pruebas y a las respuestas a las preguntas de las entrevistas realizadas a los probadores en sus entrevistas

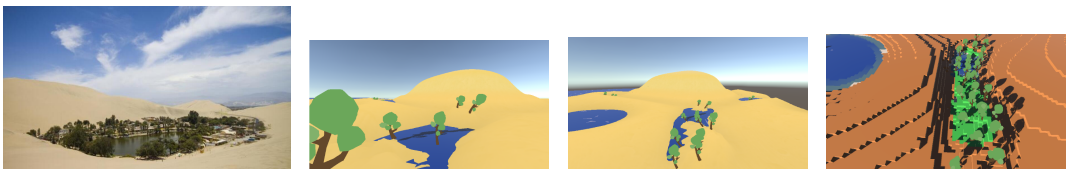
(las entrevistas completas se encuentran disponibles en Apéndice E).

	Paula	Victor	Rodrigo	Elisa	Miguel	Rocio
Entiende Parámetros BiomeObject	✓	✓	✓	✓	✓	✓
Funcionamiento Influence Range Biome Generator		✓	✓			✓
Funcionamiento Curvas de Transición Biome Generator		✓	✓			✓
Entiende Parámetros del FoliageObject	✓		✓	✓	✓	
Meter un FoliageObject en un bioma	✓	✓				
Cambiar configuración Shaders	✓		✓	✓	✓	✓
Manipular transiciones shaders	✓					
Entrar en ejecución sin ejecutar antes Generate Map		✓				✓

Tabla 6.1: Tabla De resultados a las preguntas de investigación.

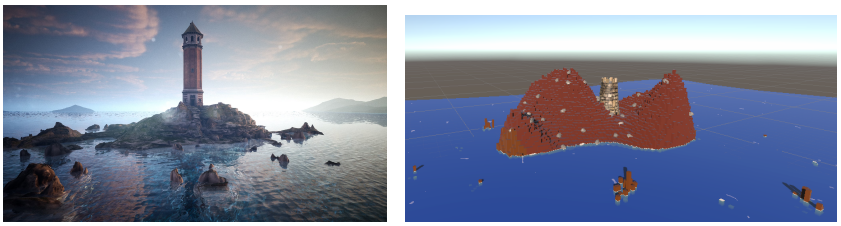
6.2.1. Resultados visuales

En esta sección se hace una comparación visual entre los terrenos creados por los probadores y las imágenes de ejemplo. Cada probador tiene su propia figura: probadora Paula (ver figura Figura 6.1), probador Víctor (ver figura Figura 6.2), probadora Elisa (ver figura Figura 6.3), probador Miguel (ver figura Figura 6.4), probador Rodrigo (ver figura Figura 6.5) y probadora Rocío (ver figura Figura 6.6).



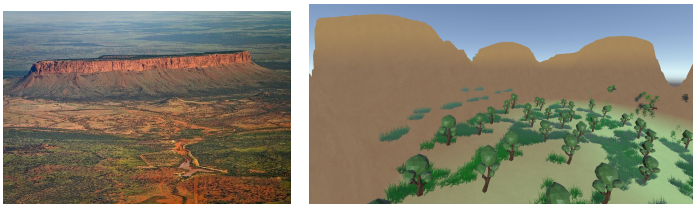
(a) Ejemplo (b) Imagen 1 Smooth (c) Imagen 2 Smooth (d) Imagen Cúbico

Figura 6.1: Comparación de los resultados de la probadora Paula



(a) Ejemplo (b) Imagen Cúbico

Figura 6.2: Comparación de los resultados del probador Víctor



(a) Ejemplo (b) Imagen Smooth

Figura 6.3: Comparación de los resultados de la probadora Elisa

6.3. Discusión

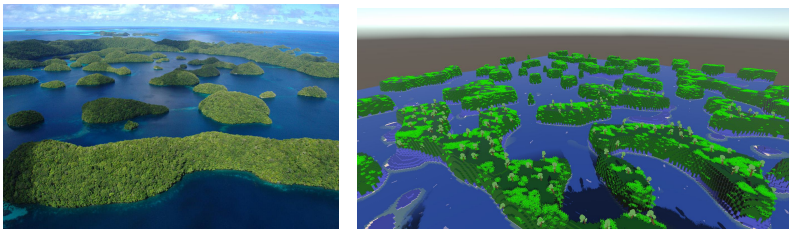
En esta sección, se interpretan los resultados obtenidos por los usuarios al interactuar con la herramienta, destacando tanto los aspectos en los que mostraron comprensión como aquellos en que afrontaron mayores desafíos. También se discuten las áreas específicas que mediante entrevistas y observaciones hemos identificado como mas problemáticas y confusas para un usuario novato en la herramienta.

6.3.1. Interpretación de resultados

Podemos afirmar que los probadores entienden la herramienta. Esto podemos afirmarlo por las imágenes de los resultados, comparándolas con las imágenes que tenían que replicar. En todos los casos el terreno obtenido guarda bastante similitud estructural con la imagen de ejemplo.

Según las respuestas obtenidas en las entrevistas (Apéndice E), a medida que los usuarios se van embarcando en el flujo de trabajo de la herramienta van aprendiendo y obteniendo agilidad manejándola. No obstante, al principio es bastante abrumador debido a la gran cantidad de parámetros que posee la herramienta.

Como podemos ver en la tabla de resultados, la inmensa mayoría de los probadores no ha manipulado los shaders en profundidad. Simplemente han cambiado los colores para ajustarlos a su escenario. Esto puede ser porque no



(a) Ejemplo (b) Imagen Cúbico

Figura 6.4: Comparación de los resultados del probador Miguel

(a) Ejemplo (b) Imagen Cubico

Figura 6.5: Comparación de los resultados del probador Rodrigo

les ha hecho falta manipular la transición (no se utiliza en los mapas cúbicos), ya que para el paisaje que estaban intentando imitar con la herramienta no era necesario, o porque no han sabido qué valores modificar para variarlo. A cambio, el usuario que los ha modificado no ha tenido problemas en entender cómo funcionaban tras unas pocas pruebas de ensayo y error.

Todos los usuarios han tenido dificultades con el uso de las curvas. En general, han resultado difíciles de interpretar. Según comentarios obtenidos en las entrevistas y la observación de los investigadores, a los probadores les surgieron muchas dudas sobre su función y en muchos casos las modificaban esperando ver los resultados de sus acciones en el terreno generado para encontrar las respuestas a sus preguntas.

Durante las pruebas era bastante común para los investigadores recibir preguntas sobre los diferentes parámetros, sobretodo los de nombres más técnicos, como los relacionados con el ruido. Son los menos intuitivos.

En cuanto a los foliageObjects los usuarios entendían todas las variables a excepción de la curva de densidad y la excepción de un probador que se confundía entre la altura random y escala random del objeto. Los biomas en general se entendían casi todas las variables a excepción de la variable weight, la cual no entendían muy bien el funcionamiento y a su vez la MeshHeight-Curve que, al igual que con weight, no entendían el funcionamiento.

(a) Ejemplo (b) Imagen Smooth

Figura 6.6: Comparación de los resultados de la probadora Rocío

6.3.2. Mejoras implementadas basadas en las pruebas de Usuarios

La mayoría de los probadores se limitó a modificar los colores del shader del terreno sin profundizar en su manipulación. Este fue el problema mas grave que pudimos detectar, que hemos afrontado mediante el uso de Tooltips. Los Tooltips son una funcionalidad que tiene Unity para añadir una pequeña descripción de cada parámetro de forma que si un usuario que esté usando la herramienta no conoce la finalidad de un parámetro específico, no tiene más que poner el ratón por encima del parámetro y aparecerá una descripción breve de como utilizar e interpretar el parámetro. De esta forma, ayudamos sobretodo a usuarios que están aprendiendo a utilizar la herramienta.

Las curvas han resultado ser el elemento mas confuso para los usuarios, quienes a menudo dependían del ensayo y error para entender sus efectos en el terreno generado, lanzando múltiples preguntas a los investigadores.
Para este problema, se han simplificado funcionalidades que requerían de múltiples curvas como las curvas que tenía la herramienta anteriormente para ajustar la transición de altura entre biomas. Tras implementar la funcionalidad, ahora sólo se dispone de una única curva para definir todo el sistema de transición entre biomas.

6.3.3. Deducciones

Aunque los usuarios a los que les hemos realizado las pruebas logran adquirir agilidad y dominio de la herramienta para moldear el terreno a su gusto, inicialmente encuentran abrumadora la gran cantidad de parámetros que se les presenta desde un inicio.

Y aunque los usuarios a los que les hemos realizado las pruebas, encuentran abrumadora la gran cantidad de parámetros que se les presenta desde un inicio, logran adquirir agilidad y dominio de la herramienta para moldear

el terreno a su gusto.

Gracias a lo visto en las pruebas, podemos afirmar que la herramienta es lo suficientemente intuitiva y compleja como para que usuarios puedan crear un mundo procedimental personalizado con terreno y vegetación.

6.3.4. Adición de funcionalidades

Durante las entrevistas les sugerimos a los usuarios funcionalidades a implementar que les gustaría que hubiese en la herramienta, estas son las aportaciones que nos sugirieron:

- Explicar todos los parámetros que se pueden manipular desde el editor con una pequeña descripción de que función tiene ese parámetro (uso de Tooltips).
- Quitar el Height de los folliageObjects ya que se puede simplificar con Scale
- Que los puntos de interés estén asociados al bioma y no a todo el mapa, además de poder mantener la generación de aquellos puntos que el usuario quiera y se generen otros en otros lados, siempre y cuando no cambie el mapa.
- Volver a generar solo determinadas zonas del mapa y poder mantener aquellas que le gusten al usuario
- Permitir que el usuario pueda pintar un mapa con colores, en el cual cada color representa un bioma de tal manera que luego se pueda generar el terreno con el mapa de biomas pintado por el usuario.

6.4. Ejemplos detallados de generación

Cabe resaltar que en las pruebas que hemos realizado con desarrolladores, se realizaban en poco tiempo, ya que queríamos probar si la herramienta se entendía y era intuitiva de usar.

Los terrenos obtenidos en las pruebas tienen la calidad esperada teniendo en cuenta el tiempo dado y los conocimientos nulos sobre la herramienta de los probadores.

En este último apartado lo que se pretende demostrar son las posibilidades que tiene la herramienta enseñando ejemplos de implementación.

Para ello, nosotros, los desarrolladores de la herramienta, hemos creados diferentes casos de uso en el que se muestran terrenos creados por nosotros mismos:

CASO 1

Para este primer ejemplo se han usado cuatro biomas principales:

- Plains: Planicies de hierba con poca variación en la altura y gran frecuencia, por lo que es el bioma que mas aparece en el terreno.
  Utiliza el sistema de vegetación para instanciar césped en gran cantidad y algunas piedras.
- Forest: Este bioma es menos frecuente y tiene asignado una altura mayor que las planicies.
  En el aparecen arboles con gran densidad, muy poca hierba y ninguna piedra.
- Mountain: Este bioma es el que tiene asignada la mayor altura de todos los biomas. No aparece ningún tipo de foliage en este bioma.
- Lake: Este bioma tiene asignada la menor altura.

Los parámetros del shader están asignados para que aparezca una textura de arena a bajas altura, una textura de césped para alturas medias tintada de verde, y para las montañas un color marrón y blanco para los valores mas altos del terreno.

Para esta configuración de terreno se han probado tres enfoques diferentes para probar la variedad en la creación de terrenos.

Estilo cartoon con mallas suavizadas, texturas difuminadas y gran cantidad de foliage de césped

Entre las texturas de arena y césped existe un mayor difuminado mientras que el difuminado entre el marrón y el blanco es mucho menor.

Terreno creado mallas cúbicas

Malla suavizada renderizado con colores planos y la difuminación entre capas al cero.

Figura 6.7: Imagen del primer terreno

Figura 6.8: Imagen del primer terreno

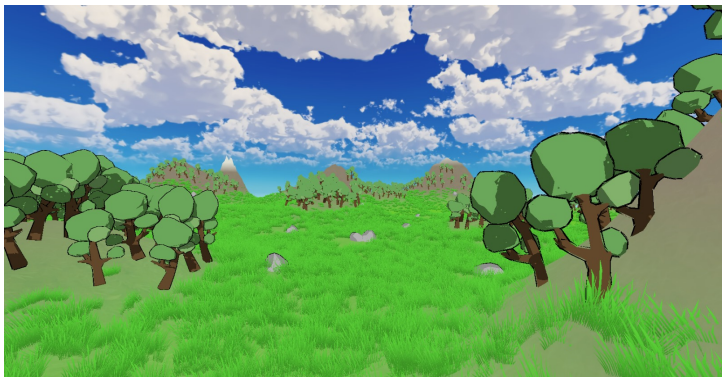


Figura 6.9: Imagen del primer terreno

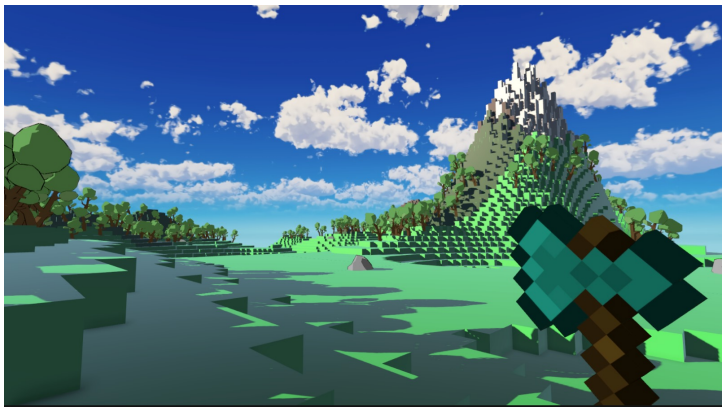


Figura 6.10: Imagen del segundo terreno

Figura 6.11: Imagen del tercer terreno

Capítulo 7

Conclusiones

Este trabajo consiste en la creación de una herramienta en Unity para desarrolladores indie, la cual permitirá la creación de terrenos procedurales de una forma sencilla e intuitiva. Estos terrenos tienen la posibilidad de albergar distintos biomas. La herramienta estará basada en componentes, usará prefabs y mapas de alturas para dejar total libertad al desarrollador a la hora de crear su propio mundo con su estética propia. De hecho, también tendrá un sistema de generación de objetos para complementar el terreno y otros con mayor importancia.

El proyecto puede encontrarse en el siguiente enlace: TFG

Todos estos objetivos han sido alcanzados durante el desarrollo de la herramienta:

1. **Diseño de un sistema que permita generar de manera procedimental un mundo 3D. Se podrá generar un terreno de estética realista, con mallas suavizadas y orgánicas, o un terreno compuesto a base de cubos**. Se ha conseguido diseñar y desarrollar el sistema y es fácil comprobar que es posible generar esos dos tipos de terrenos como evidencian los mundos de demostración creados.

   Este objetivo se ha conseguido en su totalidad, tal y como se ha detallado anteriormente.El usuario puede elegir si utilizar la herramienta para generar un terreno de mallas suavizadas o un terreno con mallas cubicas.

2. **Diseño de un sistema de *biomas* que permita distribuir regiones distintas sobre el terreno. Estos *biomas* se definen incluyendo sus características, como parámetros de ruido que determinan las características de terreno, alturas máxima y mínima y la probabilidad de que dicho bioma aparezca en el mundo.**

   Actualmente el usuario puede determinar que biomas quiere que se

generen en su mundo y a su vez puede determinar el comportamiento de dichos biomas en su totalidad por las características descritas anteriormente.

3. **Implementar un sistema intuitivo de *shaders* para personalizar los materiales a usar en cada *bioma*. Cada bioma se divide en capas, las cuales pueden tener materiales distintos, que se pueden mezclar mediante transiciones suaves**.

   El usuario en la versión actual de la herramienta no es capaz de asignar materiales a cada bioma, en cambio si puede asignar un color a cada bioma por separado, el cual es el que se usará para representar ese bioma en la visualización 2d del mapa de biomas, y en el mundo generado a partir de cubos, con el material por defecto de Unity. En cambio, en el caso del mundo liso, el usuario tiene la posibilidad de utilizar un shader personalizado que le permite determinar la textura que se mostrará en distintos niveles de altura, pudiendo hasta modificar las transiciones entre estas.

4. **Diseño de un sistema de *vegetación* que permita crear múltiples ejemplares de un objeto sobre el terreno generado, según sus *biomas*. El posicionamiento y otras formas de adaptación al entorno son configurables en la vegetación**.

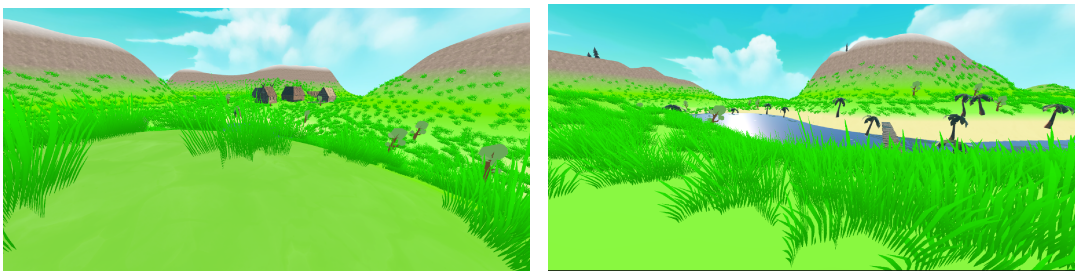
   Actualmente el sistema de generación de vegetación, posee todas los parámetros establecidos en diseño, con su respectiva funcionalidad, además hemos adaptado varias de las sugerencias de las pruebas para que la herramienta de vegetación sea mas fácil de usar y interactiva como por ejemplo el nombre de las variables Require Distance Separation y el uso de tooltips para que sea mas entendible cada parámetro.

5. **Optimización que permita manejar eficientemente mundos de gran tamaño, renderizando in-game sólo los chunks de terreno y la vegetación disponga de distintos mesh con menos vértices, pudiendo así cambiar la malla a otra de menor resolución según la distancia visible, tanto de la vegetación como de los chunks.**

   Actualmente disponemos de un sistema de generación del mundo centrado directamente en al experiencia de juego permitiendo que solo se renderizen aquellas porciones de terreno situadas en los alrededores del jugador, además tanto los terrenos como la vegetación disponen de LODS explicados anteriormente para mejorar el rendimiento.

6. **Creación de uno o varios mundo de demostración, cada uno con distintos *biomas*, *vegetación*, materiales y puntos de interés**. En total se han creado 3 mundos, uno para cada demo. Cada uno tiene su propia vegetación, puntos de interés, biomas y decoración.

- **Sample1**. Esta escena contiene el terreno con el mayor número de biomas. Su aspecto global es el de una zona boscosa con pequeños lagos o ríos. Se trata de un terreno suavizado con un aspecto cartoon, en cuyas texturas el porcentaje de tintado de color es alto. Como vegetación se ha añadido hierba y varios tipos de árboles. Como punto de interés se ha añadido un pequeño pueblo. Además como elementos extra se han añadido algunos props como un puente o un skybox de un cielo azul con nubes. Véase en: Figura C.1.
- **Sample2**. Esta escena representa un desierto. Se trata también de un terreno suavizado, sin embargo en este caso las texturas destacan más sobre los colores de tinte. Como vegetación se ha añadido a las zonas más bajas unos huesos y en el resto rocas y cactus de diversas formas. Como puntos de interés hay algunos pozos y casas en lo más alto. Para finalizar, como elemento extra se ha añadido un skybox de color rojizo. Véase en: Figura C.2.
- **Sample3**. Esta escena muestra un terreno montañoso, muy accidentado y nevado. El estilo de este mapa es cúbico. Como vegetación se han añadido rocas y una gran variedad de árboles nevados. Además, como extra se han añadido bastantes elementos rocosos de decoración, un skybox muy nubosos y una pequeña ventisca. Véase en: Figura C.3.



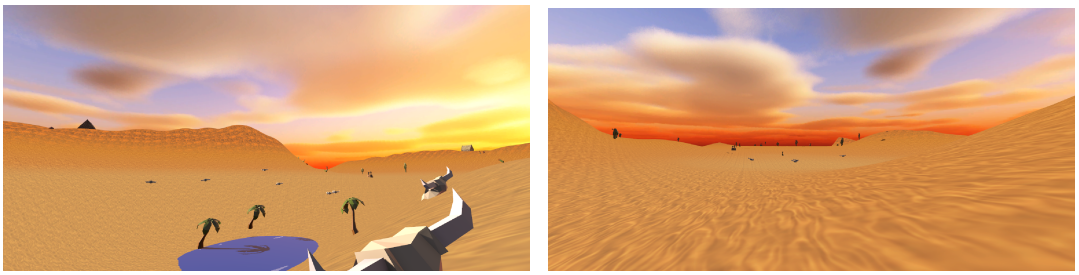
(a) Demo 1 imagen 1 (b) Demo 1 imagen 2

Figura 7.1: Imágenes de la demo 1

7. **Comprobar la eficacia y facilidad de uso de la herramienta realizando pruebas con desarrolladores reales y estudiando la facilidad y eficacia de generación de un escenario**.

7.1. Trabajo futuro

En general, continuaríamos la ampliación de la herramienta con la realimentación recibida por los usuarios a través de las pruebas, así como



(a) Demo 2 imagen 1 (b) Demo 2 imagen 2

Figura 7.2: Imágenes de la demo 2

(a) Demo 3 imagen 1 (b) Demo 3 imagen 2

Figura 7.3: Imágenes de la demo 3

incluiríamos pequeñas funcionalidades que permitan añadir mas contenido a los mundos creados.

- Permitir que el usuario pueda pintar un mapa con colores, en el cual cada color represente un bioma de tal manera que luego se pueda generar el terreno con la distribución de regiones ya pintada por el usuario. Esta funcionalidad se aplicaría solo en casos en los que se quiera realizar un terreno finito.
- Poder regenerar sólo determinadas zonas del mapa y mantener aquellas que el usuario no quiera modificar.
- Permitir la creación de ríos de dos formas, de manera pseudoaleatoria o definidos por el usuario, indicando por dónde deben aparecer y con qué forma.
- Añadir una sub-herramienta que permita definir determinados caminos entre los puntos de interés en función de las necesidades de jugabilidad del usuario.
- Poder regenerar solo aquellos puntos de interés que el usuario decida.

7.2. Reflexión final

A lo largo del proyecto, se han logrado integrar diversas técnicas como la generación de terrenos, biomas, shaders y vegetación, enfrentando retos y problemas que surgían por el camino, pero al final de todo, podemos decir que estamos orgullosos del trabajo realizado.
Este proyecto ha ampliado nuestros conocimientos y habilidades, y esperamos que la herramienta sea un recurso valioso para la comunidad de desarrolladores indie.

Apéndice A

Contribuciones individuales

En este apéndice se enumeran las tareas realizadas por cada uno de los integrantes para llevar a cabo el proyecto. Cabe destacar que, en ciertas tareas ha habido colaboración mutua debido a su complejidad.

En nuestro repositorio de Github pueden encontrarse todas las tareas realizadas a lo largo de todo el proyecto, incluyendo por qué integrante ha sido realizada.

A.1. Ignacio del Castillo Rubio

Las contribuciones al proyecto se dividen en dos partes, por un lado la herramienta y otro la memoria.

A.1.1. Aportaciones para la creación de la Herramienta

- Modificar el proceso en el que se sitúan y conectan los vértices formando triángulos con el objetivo de generar terrenos suavizados, en vez de cúbicos.
- Primera implementación del sistema de vegetación, para posicionar objetos sobre el terreno dados unos parámetros básicos como la densidad
- Creación del prototipo visual de la herramienta. En este prototipo inicial se diseño e implementó la interfaz en un, que aspecto tendría la herramienta, en un estado acabado.
  Este prototipo se realizó a partir de un proyecto vacío de Unity aparte, teniendo en cuenta los siguientes objetivos:
  - Poder tener una idea mas clara de como funcionaría la herramienta para saber como estructurar la herramienta por dentro y como se comunicarían las clases entre sí.

    - Planificar mejor el trabajo necesario para conseguir dicha funcionalidad.
    - Que tipos y cuantos tipos de scriptableObjects se necesitarían para abordar toda la funcionalidad de vegetación, biomas y demás elementos.
    - Cuantos parámetros, su función y de que forma se modificarían para generar toda la funcionalidad propuesta definida en los objetivos de la herramienta. Esto incluye la funcionalidad tanto del generador de terreno como de los scriptableObject

- Implementación del editor modificable para los scriptableObjects, es decir, la lógica interna para personalizar el funcionamiento y comportamiento de los parámetros para mejorar la experiencia de usuario lo mas posible (esconder variables que no son necesarias en ciertos casos, elementos de interfaz como deslizadores para números con rangos)

- Sistema de personalización del césped para usar con la herramienta vegetación en el que se puede configurar la textura, forma y gradiente del césped.

- Sistema que permite el posicionamiento de un objeto jugador, especificado por el usuario cada vez que se genera un nuevo terreno.
  Permite elegir el bioma en el que se quiere posicionar al jugador y ajustes relacionados con la aleatoriedad que utiliza la funcionalidad para generar una posición aleatoria.

- Shader personalizado que separa el terreno por capas de alturas :

    - Sistema para dividir la malla en capas por alturas y renderizar la malla del terreno según estas capas
    - Cambiar la textura de cada una de las capas y modificar la escala de cada una
    - Combinar un color con cada una de las texturas de las capas y elegir si afecta mucho o poco a el color a la textura
    - Para cada una de las capas de altura, tener la posibilidad de combinar los colores de cada capa con el resto de las capas contiguas, controlando con un valor la mezcla entre una capa y el resto
    - Documentación del funcionamiento del shader, explicando que hace cada parte del código y como es la estructura de un shader URP en lenguaje Hlsl

A.1.2. Memoria

- Discusión de los resultados obtenidos en las pruebas realizadas

- Mejoras implementadas basadas en las pruebas de usuarios
- Conclusión de las pruebas de usuario
- Reflexión final
- Desarrollo de la herramienta
- Renderizar mallas con múltiples texturas y colores dependiendo de la altura
- Posicionamiento del jugador en un bioma
- Manual de uso en las guías del usuario
- Iteraciones y generación de ideas

A.2. Javier Comas de Frutos

Las contribuciones al proyecto se dividen en dos partes, por un lado la herramienta y otro la memoria.

A.2.1. Aportaciones para la creación de la Herramienta

- Adición del punto de partida.
- Creación de mallas suavizadas y más lisas para la generación de terrenos, para tener terrenos más lisos en vez de sólo cúbicos.
- Funcionalidad de alturas y tamaño configurable de bloques del tipo Cúbico.
- Sistema de eliminación de mapas antiguos, guardado (en la escena) de los mapas deprecados con la hora en la que se han eliminado.
- Creación de NoiseSettings para la configuración del ruido y encapsular los parámetros de ruido.
- Sistema de vegetación.
  - Creación del sistema de vegetación.
  - Scripteable Object de foliage (vegetación).
  - Adaptación de los objetos al terreno.
  - Integración con el resto de componentes de la herramienta.
- Perlin Worms (Para la generación de caminos/ríos por el mapa)

- Primera versión del Algoritmo *PoissonDiscSampler* el cual iba a ser usado junto con el perlin Worms para la creación de ríos y caminos.
- Configuración del editor, aspecto visual.
  - MapGenerator: De cara a que el usuario le sea mas fácil, visual y rápido encontrar los parámetros que desea cambiar, ya que este componente dispone de muchos parámetros.
  - FoliageObject: Al igual que el MapGenerator, como este scripteableObject dispone de muchos parámetros, ordenador y poner títulos facilita al usuario encontrar y modificar la configuración deseada.
- Densidad de cada bioma / porcentaje de aparición de cada bioma
- Refactorización de la clase Chunks.
- Refactorización de creación de las Mesh.
- Arreglo de los vértices con las paredes exteriores de cada chunk (Cúbico), tras la refactorización de los chunks se produjo un bug en el cual los últimos vértices pertenecientes a la malla de cada chunk no se creaban, el arreglo este consistió en solucionar la delimitación de cuantas casillas de ruido pertenecían a cada chunk.
- Arreglo del instanciamiento de objetos con el endless terrain.
- Creación de los planes / objetivos para las pruebas.
  - Eliminación de usings innecesarios.
  - Eliminación de comentarios innecesarios.
  - Eliminación de parámetros innecesarios de diversas funciones en los scripts.
  - Añadidos y corregidos comentarios en el código.
  - Cambio de partes del código que no cumplían las reglas semánticas. Esto incluye cambio de la primera letra de las funciones a mayúscula y cambio del nombre de algunas variables para hacerlas más intuitivas.
- Documentación para la Herramienta en el readMe.md de el repositorio de Github.
- Adición de diversos objetos destinados para la versión de las pruebas, provenientes de paquetes de la asset Store.

A.2.2. Memoria

- Resumen
- Abstract
- Capítulo 2. Herramientas, Generación Procedimental, Foliage y Objetivos
- Capítulo 4. Metodología. Subapartado de Scrum y parte del subapartado de Herramientas
- Capítulo 5. Análisis, Diseño y Implementación
- Capítulo 6. Pruebas, Resultados y Discusión
- Capítulo 7. Conclusión, Trabajo Futuro, Conclusión (Inglés)
- Apéndice D. Manual de Uso
- Apéndice E. Entrevista
- Diversas imágenes de los apartados expuestos anteriormente y referencias biográficas

A.3. Sara Isabel García Moral

Las contribuciones al proyecto se dividen en dos partes, por un lado la herramienta y otro la memoria.

A.3.1. Aportaciones para la creación de la Herramienta

- Buscar personaje en 1ª persona que se mueva por el escenario e importar el paquete al proyecto.
- Prefab de la torre del punto de interés castillo, esto incluye modelado de la malla 3D de una torre en Blender y búsqueda de una textura de ladrillos.
- Incorporación del componente *LOD Group* de Unity a los chunks del mapa generado y su configuración correspondiente. Para ello se ha modificado a constructora de la clase *Chunk*. En ella también se han ajustado los colliders de acuerdo a los LODs.
- Método que calcula en tamaño de los chunks en el script *MapGenerator*.
- Shader de simulación de viento y aplicación de este los prefabs de hierba y árboles como vegetación.

- Script *Endless Terrain*, con el algoritmo para generar y activar los chunks cercanos al jugador y desactivar los que están más allá de su máxima distancia de visión, para lo que se han creado los métodos Update(), SetVisible() e IsVisible() del script *Chunk*. También se han añadido las condiciones de Endless Terrain en el script de *Map Generator*, si esta activado no se genera el mapa al dar al botón de generar.
- Script de *PoissonDiscSampler*, que implementa el algoritmo de *PoissonDisc* para crear los puntos de interés. Modificación de este para ampliar los puntos de interés con la condición de generación en un rango de altura y unos biomas concretos.
- Método Generate() del script de *InterestPoint*, que se encarga de colocar los puntos de interés en el mapa adecuadamente.
- Aportación sobre el Endless Terrain, los puntos de interés y WindShader en el manual de la herramienta.
- Añadidas Tooltips en los scripts visuales desde el editor: Biome, Foliage, InterestPoint, BiomeGenerator, NoiseSettings y MapGenerator.
- Proyecto con con las demos en la carpeta "Demos" del proyecto. Dentro de el proyecto de demos añadidas 3 demos, separadas en 3 escenas. Una con varios biomas forestales y pueblos, el segundo imitando un desierto, y el tercero es un escenario de hielo y nieve.
- Refactorización en varios scripts del proyecto:
  - Eliminación de usings innecesarios.
  - Eliminación de comentarios innecesarios.
  - Eliminación de parámetros innecesarios de diversas funciones en los scripts.
  - Añadidos y corregidos comentarios en el código.
  - Cambio de partes del código que no cumplían las reglas semánticas. Esto incluye cambio de la primera letra de las funciones a mayúscula y cambio del nombre de algunas variables para hacerlas más intuitivas.

A.3.2. Memoria

- Portada.
- Capítulo 1. Completo.
- Capítulo 2. Subapartado de Videojuegos de la Investigación y subapartado de Unreal Engine de Herramientas de la Investigación. Descripción

de los LODs en el subapartado de generación de mallas de la Generación Procedimental. Apartado de Endless Terrain. Apartado de Puntos de Interés. Apartado de Shaders.

- Capítulo 3.
- Capítulo 4.
- Capítulo 5. Análisis e introducción de cada sección. LODs, Endless Terrain, simulación de viento y puntos de interés en Diseño e Implementación.
- Añadidas al capítulo del manual las secciones de los puntos de interés, el Endless Terrain y WindShader.
- Apéndice A. Este apartado de las contribuciones y la introducción del apartado.
- Apéndice B. Introduction
- Apéndice E. Entrevista de Paula y Rodrigo.
- Imágenes:
  - Imágenes referentes a los LODs (level-of-detail.jpg y LODs.png).
  - Imagen referentes al Endless Terrain (endless.png, endleesExample.png, ejemplo1endless.png y ejemplo2endless.png).
  - Imagen del mapa de Read Dead Redemption 2 (rdr2.png).
  - Imagen de la versión de Unity (version-unity.png).
  - Imágenes referentes al shader de viento (wind-shader.png y color-shader.png, paso1.png, paso2.png y paso3.png).
  - Imagen del cálculo del tamaño de casilla para el algoritmo de PoissonDisc (pitagoras.png).
  - Imagen de la configuración en el editor de Unity de un Interest-Point (interest-point.png).
  - Imágenes de los biomas de ejemplo de las pruebas (Bioma1.jpg, Bioma2.png, Bioma3.png, Bioma4.jpg, Bioma5.jpg, Bioma6.jpg, Bioma7.jpg y Bioma8.png).
  - Imágenes de los resultados de las pruebas (Desierto1.png, Desierto2.png, Desierto3.png, Elisa.png, mikel.jpg, Rocio.png, Rodrigo.png y victor.png).

A.4. Javier Enrique Villegas Montelongo

Las contribuciones al proyecto se dividen en dos partes, por un lado la herramienta y otro la memoria.

A.4.1. Aportaciones para la creación de la Herramienta

- Arreglo de fallos relacionados con el terreno liso y el tamaño de los chunks.
- Refactorización de gran parte del código para poder añadir la funcionalidad de biomas:
  - En MapGenerator, quitado el array bidimensional "cellMap" en el que cada "cell" contenía la información entera de su posición, a favor de una variable de tipo "MapInfo", que guarda el mapa de alturas, el mapa de ruido, y en un futuro los biomas y su influencias por cada punto del mapa.
  - Además, hubo que mover parte del código ya existente de este script a otros scripts que se crearon con la introducción de los biomas.
  - Planteado de nuevo el orden en el que se ejecutan los pasos necesarios para generar el mapa, dado que ahora no es el mapa quien controla la altura de cada punto, sino los nuevos biomas influyentes.
- Implementación del sistema de biomas:
  - Diseño, creación e implementación del script "BiomeGenerator" encargado de generar la posición, forma de cada bioma incluido en una lista configurada por el usuario en el mapa final, además calcular que tan influyente es cada bioma en cada posición del mapa.
  - Diseño, creación e implementación del script "Biome", que contiene varios atributos ajustables por el usuario, y además genera su propio mapa de ruido, que luego "MapGenerator" utilizando las influencias de cada uno calculadas desde "BiomeGenerator" se vean reflejadas en el mapa.
  - A la hora de crear la textura a color del mapa generado, ahora el color elegido para cada punto viene determinado por el bioma más influyente en ese punto.
- Cálculo de divisores del tamaño de chunk para poder hacer distintos niveles de detalle sin que haya huecos entre los chunks.
- Diseño y preparación de las pruebas con usuarios:
  - Diseño del plan de pruebas.
  - Preparación del entorno en el que se hicieron las pruebas.
  - Búsqueda de las imágenes utilizadas para las pruebas de usuarios.

    - Diseño del guión de la entrevista posterior.

- Cambios relacionados con los resultados de las pruebas:

    - Quitadas las curvas de "NoiseTransition" y "HeightTransition" de MapGenerator además del atributo "InfluenceRange" de BiomeGenerator, a favor de una sola curva "BiomeTransition" en BiomeGenerator a partir de la cual se gestiona todo lo relacionado con las transiciones de los biomas.

- Optimización y arreglos del código:

    - Sustituido varios mapas de ruido y de altura para que en vez de que sean arrays bidimensionales de números en punto flotante, sea un número del 0 al 255 (un byte) y se transformen a la hora de devolverlos, haciendo que esos mapas ocupen 4 veces menos en memoria.
    - Quitadas copias de valores grandes innecesarias, como el mapa de influencias.

A.4.2. Memoria

- Capítulo 2. Apartado de Biomas en la sección de Generador de terrenos.
- Capítulo 3. Interfaz del editor, parte de biomas.
- Capítulo 5. Los Sub-apartados de LODs y Biomas de la Implementación
- Capítulo 6. Tabla de Resultados.
- Apéndice E. Entrevista a Elisa, Miguel y Rocío.
- Imágenes:

    - Creada y añadida imagen de la visualización de los LODs en el mapa (LODs.png).
    - Creada y añadida imagen del diagrama simplificado de clases principales (DiagramaClases.png).
    - Creadas las imágenes del planteamiento de la interfaz de varios puntos de la herramienta en la ventana de inspector de Unity (Capítulo 3, figuras 3.1, 3.2, 3.3 y 3.4)

Appendix B

Introduction

Procedural generation is the method of creating content using algorithms instead of manually. Depending on the context in which it is used, its purpose is different. As this is a Final Degree Project belonging to the Video Game Development Degree, our context is video games. The main application of this technique in this context is the creation of content such as scenarios, missions or game objects.

The algorithms commonly used to carry out this type of techniques are usually based on nodes, cellular automata, or branching and pruning. These types of algorithms are usually recursive (also called recurrent). Also popular are noise algorithms, which produce coherent and smooth pseudo-random patterns, providing a good basis for modeling natural variations, e.g. in topography (e.g. Perlin noise and Simplex noise).

The procedural approach greatly reduces the time to generate a virtual world because the process of shaping the map and placing elements such as trees is fully automated. It also increases the variability of worlds and their randomness, which can increase their realism. Furthermore, if done at runtime, it provides unique experiences for each player.

It is especially useful in open-world games where exploration is an integral part of the player's experience. It is worth noting that this type of game has become particularly important in the last decade due to the great success of titles belonging to this genre such as *Grand Theft Auto V* Rockstar North (2013), *Red Dead Redemption 2* Rockstar Studios (2018) and *The Legend of Zelda: Breath of the Wild* Nintendo EPD (2017). All of them are built on immense virtual worlds with great possibilities for exploration and interaction.

B.1. Motivation

Many independent developers (or *indies*) who use Unity do not have a powerful tool that allows them to generate terrains in a procedural and fully customisable way. In many cases, the tools available in the Unity Asset Store are paid tools that allow them to create very specific and generally poor types of worlds.

Many developers end up creating their own algorithms and data structures to generate the scenery for their games. They use the same training resources and, as a consecuence, the results often look very similar to each other.

B.1.1. Purpose

The purpose of this work is to contribute to the existence of development tools that facilitate the procedural generation of terrain and vegetation in video games in Unity. The aim is to reduce the workload and investment of effort that small developers need to carry out to build their own procedural generators of virtual worlds.

The profile of our potential customers is the independent developer interested in procedurally generated virtual worlds. It is true that there are already tools for the creation of terrains, however, there are hardly any tools on the market that create complex, varied virtual worlds, with biomes and that are highly customisable in a direct and simple way. For this reason we have chosen to create a tool that allows us to generate terrains and vegetation in a procedural way using an editor, a set of prefabs and height maps among other basic resources.

The tool is an extension of the most widely used videogame development environment today, which is *Unity*. Technologies (2024). Worldwide 73% of development studios with less than 50 employees use it and, specifically in Spain, 85% of companies use it, compared to the second most used engine, which is Unreal Engine, with 23%, according to the *White Book of Videogames*. de Empresas Desarrolladoras de Videojuegos y Software de Entretenimiento) (2022)

The main functions we are looking for with this proposed framework are simplicity and variability. To illustrate the possibilities of this framework, once developed, we created several game prototypes using the proposed software.

B.1.2. Scope

The current landscape of tools for the creation of procedural worlds presents considerable complexity. A thorough analysis by an independent developer is required to fully understand the operation and effective imple-

mentation of these tools. The main objective of this project is to offer a solution that allows these developers to avoid this work of technical understanding of the tools in question.

Our proposal is based on minimising this workload, seeking to provide a simple interface and clear functionalities that allow the independent developer to start generating worlds procedurally with a minimum investment of effort. By facilitating the integration of all the functionalities required for the development of your game, we offer a fast and straightforward experience. These virtual world generation functionalities have been meticulously tuned to ensure harmony and cohesion.

B.2. Related Subjects

The following is a description of the relationships of this dissertation with respect to different subjects taken during the degree course.

The use of different Artificial Intelligence (AI) algorithms for terrain creation and modelling, such as Perlin's noise, means that the subject of Artificial Intelligence for Video Games the subject most closely related to this work. These algorithms are based on recursion, so it is related to Discrete Mathematics as well.

The design of both the tool and the example prototype, as well as the development of the Game Design Document (GDD) is supported by the Video Game Design subject.

The production of the project, organisation and management of the repository and tasks are based on Agile Production Methodologies.

The subject Video Game Engines is also related to the project due to the continuous use of the Unity development environment.

The development of our tool has been significantly influenced by the understanding and practical application gained through subjects such as Data Structure and Algorithms and Algorithmic Methods in Problem Solving. These subjects have been fundamental pillars in our ability to tackle the algorithmic and structural challenges inherent to this project.

The Degree in Video Game Development has a list of subjects related to programming that, although not directly related to the subject of the work, are essential, as they have provided us with programming knowledge and the ability to design software architectures and structure and solve technical problems properly. These subjects are Programming Fundamentals, Video Game Programming Technology and Computer Graphics.

In addition, the different Project subjects (I, II and III) and Company Internships have provided skills in order to work in a team and develop a project like this one, which is more ambitious and takes up a longer period of time.

B.3. Report Estructure

The structure of this project consists of the following sections:

- **chapter 1: Introducción.** The proposal, the scope, the purpose of this work and the degree subjects related to this project are presented.
- **chapter 2: Estado de la cuestión.** It reviews the state of the art in AI used within video games or as a technique to aid in their development.
- **chapter 3: Objetivos y especificación.** The main objectives of the work are presented and a specification is made of all the requirements necessary to be able to develop the proposed tool.
- **chapter 4: Metodología.** It explains the methodology used to develop the project and what specific tools have been used in the process.
- **chapter 5: Desarrollo.** A detailed explanation of the analysis, design and implementation of our project is given.
- **chapter 6: Resultados.** It begins with an explanation of the tests and the objectives. Subsequently, the results obtained from the tests with real users that we have carried out to validate the functioning of the tool and evaluate its usefulness for real developers are discussed.
- **chapter 7: Conclusiones.** The conclusions generated as a result of the comparison of the final version of the tool with the objectives defined at the beginning are shown in order to validate whether these have been met.

Appendix C

Conclusions

This work consists of the creation of a tool in Unity for indie developers, which will allow the creation of procedural terrains in a simple and intuitive way. These terrains have the possibility to host different biomes. The tool will be component-based, will use prefabs and height maps to give developers the freedom to create their own world with their own aesthetics. In fact, it will also have a system for generating objects to complement the terrain and others of greater importance.

The project can be found at the following link: TFG

These objectives have been achieved during the development of the tool:

1. **Design of a system to procedurally generate a 3D world. It will be possible to generate a terrain with a realistic aesthetic, with smoothed and organic meshes, or a terrain composed of cubes**. The system has been designed and developed and it is easy to verify that it is possible to generate these two types of terrain, as evidenced by the demonstration worlds created.

   This objective has been fully achieved, as detailed above, and the user can choose whether to use the tool to generate a smoothed mesh terrain or a cubic mesh terrain.

2. **Design of a system of *biomes* which allows for the distribution of different regions on the ground. These *biomes* are defined including their characteristics, such as noise parameters that determine the terrain characteristics, maximum and minimum heights and the probability of such a biome appearing in the world.**

   Currently the user can determine which biomes he wants to be generated in his world and can determine the behaviour of these biomes as a whole by the characteristics described above.

3. **Implement an intuitive system of *shaders* to customise the**

materials to be used in each *biome*. Each biome is divided into layers, which can have different materials, which can be mixed using smooth transitions.

The user in the current version of the tool is not able to assign materials to each biome, but can assign a colour to each biome separately, which will be used to represent that biome in the 2d visualisation of the biome map, and in the world generated from cubes, with Unity's default material. On the other hand, in the case of the smooth world, the user has the possibility of using a custom shader that allows him to determine the texture that will be displayed at different levels of height, and can even modify the transitions between them.

4. **Design of a vegetation system that allows for the creation of multiple copies of an object on the generated terrain, according to its *biomes*. Positioning and other forms of adaptation to the environment are configurable in the *vegetation***.

   Currently the vegetation generation system has all the parameters established in the design, with their respective functionality. We have also adapted several of the suggestions from the tests to make the vegetation tool more user-friendly and interactive, such as the name of the variables Require Distance Separation and the use of tooltips to make each parameter more understandable.

5. **Optimisation to efficiently handle large worlds, rendering ingame only terrain chunks and vegetation in different meshes with fewer vertices, so that the mesh can be changed to a lower resolution mesh depending on the visible distance of both vegetation and chunks.**

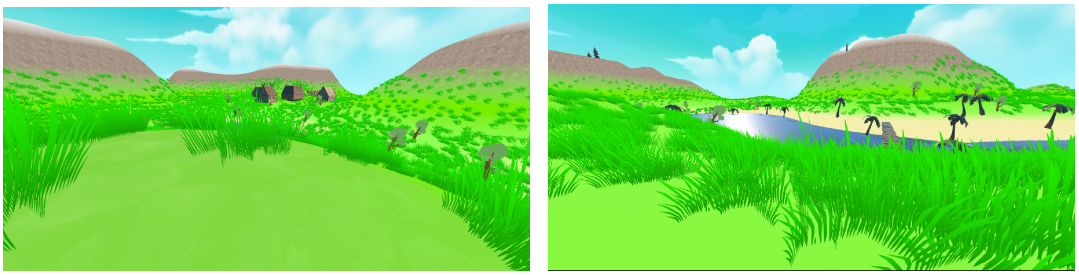
   We currently have a world generation system focused directly on the game experience allowing only those portions of terrain located in the player's surroundings to be rendered, and both terrain and vegetation have LODS explained above to improve performance.

6. **Creation of one or more demo world(s), each with a different *biomes*, *vegetation*, materials and points of interest**. In total 3 worlds have been created, one for each demo. Each has its own vegetation, points of interest, biomes and decoration.

   - **Sample1**. This scene contains the terrain with the largest number of biomes. Its overall appearance is that of a wooded area with small lakes or rivers. It is a smoothed terrain with a cartoon-like appearance, with a high percentage of colour tinting in the textures. Grass and various types of trees have been added as vegetation. A small village has been added as a point of interest.

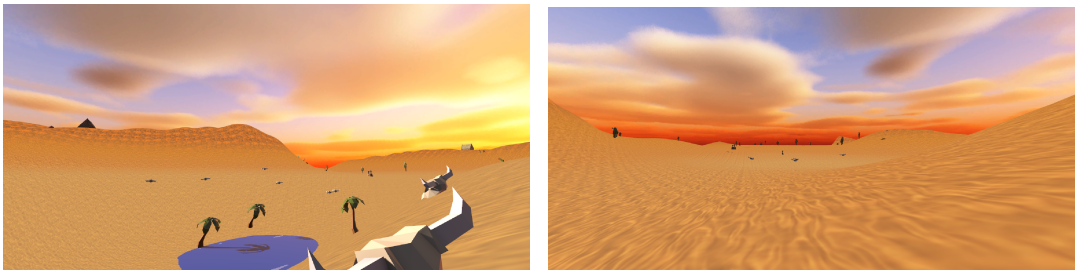
Also as extra elements some props have been added like a bridge or a skybox of a blue sky with clouds. See in: Figure C.1.

- **Sample2**. This scene represents a desert. It is also a softened terrain, but in this case the textures stand out more than the dyed colours. As vegetation, bones have been added to the lower areas and rocks and cacti of various shapes in the rest. As points of interest there are some wells and houses at the top. Finally, a reddish skybox has been added as an extra element. See in: Figure C.2.
- **Sample3**. This scene shows a mountainous, very rugged and snowy terrain. The style of this map is cubic. As vegetation, rocks and a variety of snowy trees have been added. Also, as an extra, quite a few decorative rocky elements, a very cloudy skybox and a small blizzard have been added. See in: Figure C.3.



(a) Demo 1 image 1 (b) Demo 1 image 2

Figure C.1: Images from demo 1



(a) Demo 2 image 1 (b) Demo 2 image 2

Figure C.2: Images from demo 2

7. **Test the effectiveness and ease of use of the tool by testing with real developers and studying the ease and effectiveness of generating a scenario.**.

(a) Demo 3 image 1 (b) Demo 3 image 2

Figure C.3: Images from demo 3

C.1. Future work

In general, we would continue to expand the tool with feedback received from users through testing, as well as include small features that allow more content to be added to the worlds created.

- Allow the user to paint a map with colours, in which each colour represents a biome, so that the terrain can then be generated with the distribution of regions already painted by the user. This functionality would apply only in cases where a finite terrain is desired.
- To be able to regenerate only certain areas of the map and keep those that the user does not want to modify.
- Allow the creation of rivers in two ways, pseudo-random or user-defined, indicating where they should appear and in what shape.
- Add a sub-tool that allows to define certain paths between points of interest according to the user's gameplay needs.
- Being able to regenerate only those points of interest that the user chooses.

C.2. Final Thought

Throughout the project, we have managed to integrate various techniques such as terrain generation, biomes, shaders and vegetation, facing challenges and problems along the way, but at the end of the day, we can say that we are proud of the work we have done.

This project has expanded our knowledge and skills, and we hope that the tool will be a valuable resource for the indie developer community.

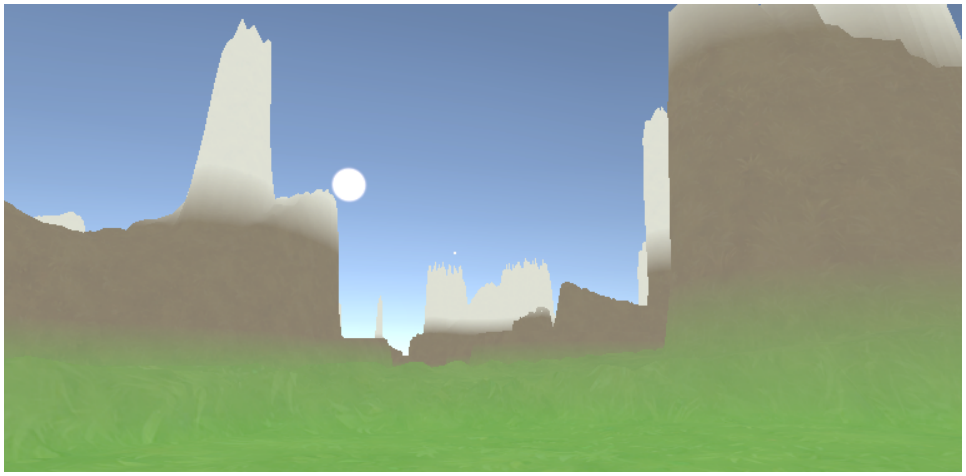
Apéndice D

Manual de uso

La documentación actualizada de la herramienta se encuentra actualizada en todo momento en :
Repositorio de Github

D.0.1. Badges

- Movimiento y cámara de ejemplo para moverse por el mapa:modular-first-person-controller
- Unity Version: 2022.3.1f1 LTS
- Animales
- Árboles



D.0.2. Descripción y contexto

Procedural OVO Worlds es una herramienta de creación de mapas procedurales 3D, es una potente solución que permite generar sus propios mapas personalizados con una gran variedad de estilos y elementos. La herramienta proporciona dos estilos principales: **Smooth y Cúbico**(tipo Minecraft), proporcionando así opciones visuales diversificadas para satisfacer las preferencias de los usuarios.

En cuanto a la generación y estilos de terrenos, esta es capaz de generar una amplia gama de estilos, permitiendo así adaptarse a múltiples y diversas temáticas y necesidades creativas del usuario. Los tipos de terreno que genera son biomas y estos determinan las características de este tipo de terreno. Por otro lado la herramienta dispone de dos sistemas de generación de objetos, el sistema de generación de puntos de interés el cual sirve para generar objetos que se les quiere dar una mayor importancia y por otro lado el sistema de vegetación para poblar el mapa generado con estos objetos.

D.0.3. Guía de instalación

Requisitos Previos:

- Tener instalado Unity cuya versión sea igual o superior a 2022.3.1f1 LTS.
- Tener descargado el paquete Procedural OVO Worlds.

Pasos a Seguir:
1. Abrir el Proyecto en Unity:

- Iniciar Unity Hub.
- Selecciona el proyecto en el que deseas importar el paquete o crea uno nuevo si es necesario.

2. Importar el Paquete:

- En la interfaz de Unity, ve al menú “Assets”.
- Selecciona “Import Package” >“Custom Package”.
- Navega hasta la ubicación donde descargaste el paquete.
- Selecciona el paquete y haz clic en “Abrir”.

3. Confirmar los Elementos a Importar:

- Aparecerá una ventana que muestra todos los elementos que se importarán con el paquete.

- Puedes desmarcar elementos que no desees importar en este momento (Nota: puede afectar a ciertos elementos).

4. Importar los Elementos:

- Haz clic en "Import" para comenzar a importar los elementos seleccionados.
- Espera a que Unity complete el proceso de importación.

5. Verificar la Importación:

- Una vez completada la importación, verifica que todos los elementos se hayan importado correctamente.

D.1. Guía de usuario

En primer lugar encontramos el prefab **Map Generator**, en este observamos el **componente principal** de la herramienta que es el ***Map Generator*** (ver en Figura D.1).

Basics Elements:

- **DrawMode**: Indica el tipo de mapa que se va a generar, este puede ser
  - NoiseMap: mapa 2D en blanco y negro, que representa el ruido de Perlin
  - ColorMap: mapa 2D a color
  - CubicMap: mapa 3D cúbico, al estilo minecraft
  - CartoonMap: mapa 3D smooth, terreno suavizado
- **GameObject Map3D**: Es el gameobject padre en el cual se va a generar todo el mapa, incluyendo objetos.
- **MapSize**: Tamaño del mapa que se va a generar
- **Seed**: Este atributo es un entero que se utiliza para inicializar el generador de números aleatorios utilizado por Unity para la creación del mapa ruido de distribución de los biomas por el mapa. Al definir este valor, se especifica el valor inicial de la semilla del generador de números aleatorios, lo que, a su vez, afecta visualmente a la apariencia del ruido creado en la escena. Definiendo de forma diferente este valor, se pueden obtener una variedad de vistas del ruido creado.
- **Offset**: La propiedad "offset" representa el desplazamiento del ruido generado, este desplazamiento permite ajustar la posición inicial del ruido en la escena. Este es un vector2 que especifica la cantidad de desplazamiento en las direcciones horizontal y vertical.

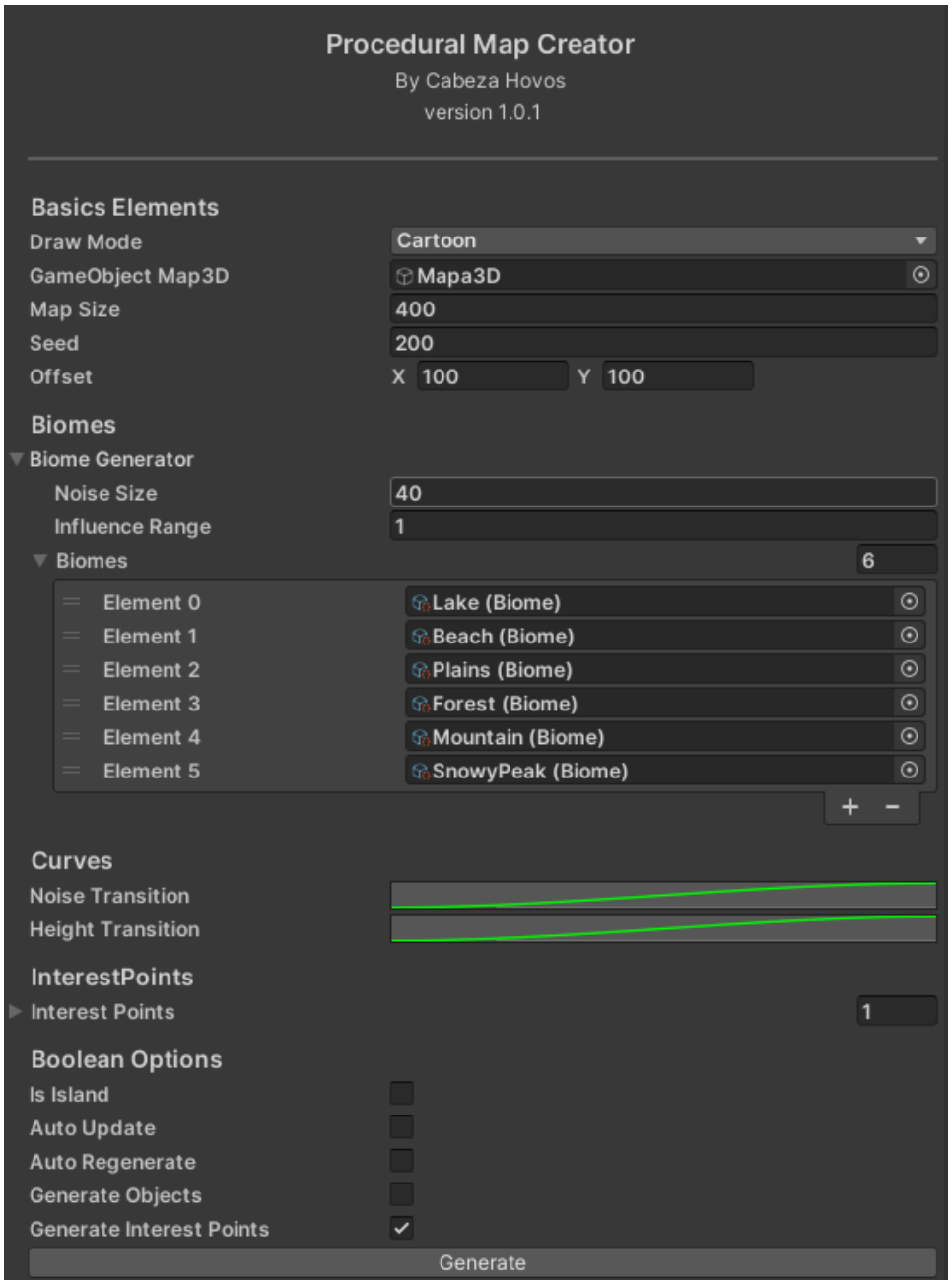


Figura D.1: Map Generator Script Editor

- **Biomes**: BiomeGenerator es el encargado de la distribución de forma dinámica de los biomas por el mapa utilizando un mapa de ruido cuya semilla es la que se ha establecido anteriormente. En este encontramos:
  - **Noise Size**: El tamaño del ruido del mapa utilizado para generar los biomas, a más pequeño, mas pequeños serán los biomas en general.
  - **Biome Transition**: Una curva de animación de Unity que se utiliza para determinar cómo manejar la transición de las características de un bioma a otro.
  - **Biomes**: Conjunto de biomas que van a conformar el mapa. El orden en el que se colocan es un factor a tener en cuenta, ya que cada bioma aparecerá junto a los biomas que tenga contiguos en este array.
- **Interest Points**: Lista de los puntos de interés que aparecerán en el mapa, solo si el check de "Generate Interest Points" está marcado. Deben asociarse "Interest Point Objects". Los puntos de interés permiten colocar los prefabs que se deseen de manera pseudoaleatoria por el mapa. **Nota**: cada vez que se genere un mapa, los puntos de interés se volverán a generar, al ser aleatorio, su ubicación variará.
- ***Boolean Options***
  - **IsIsland**: Booleano que permita que el mapa generado tenga forma de isla
  - **Auto Update**: Booleando que permite que cualquier cambio realizado en este componente se actualice directamente creando un mapa nuevo instantáneo
  - **Generate Objects**: Booleando que permite la creación de gameobjects en el mapa
  - **Generate InterestPoints**: Booleando que permite la creación de los puntos de interés anteriormente establecidos en el mapa

D.1.1. Biome Object

Este objeto determina el comportamiento que va a tener el terreno de su bioma, para ello, dicho comportamiento se establece con los parámetros que se pueden ver en: Figura D.2

- ***Noise Settings***
  - **Octaves**: El número de octavas utilizadas en el algoritmo de ruido. Cada octava es una capa de ruido que se suma al resultado final. A medida que se agregan más octavas, el ruido generado se vuelve más detallado

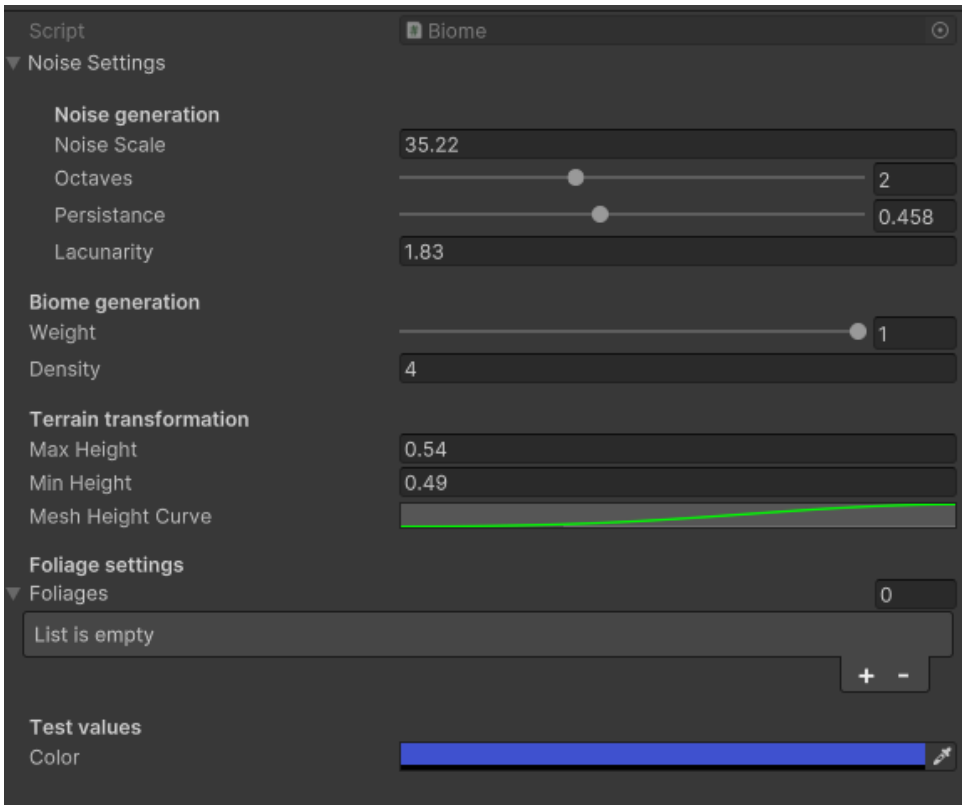


Figura D.2: Biome Object

- **Persistance**: La persistencia controla la amplitud de cada octava. Un valor más bajo reducirá el efecto de las octavas posteriores
- **Lacunarity**: Un multiplicador que determina cómo de rápido aumenta la frecuencia para cada octava sucesiva en una función de ruido de Perlin

- ***Biome Generation***
  - **Density**: Número que representa cuánto va a ocupar el bioma en el mapa total con respecto a otros biomas

- ***Terrain Transformation***
  - **Max Height**: Altura máxima del terreno que puede llegar a generarse el bioma
  - **Min Height**: Altura mínima del terreno que puede llegar a generarse el bioma

- ***Foliage Settings***
  - **Foliages**: Conjunto de objetos que se pueden generar en el bioma

- ***Color***
  - **Color**: Color con el que se representará el bioma cuando se crea en modo cúbico

D.1.2. Foliage Object

Este objeto (ver en Figura D.3), permite la instanciación de elementos a lo largo del mapa generado. Permite definir las características y parámetros de un punto de interés. Para crear un objeto de este tipo se deben seguir los siguientes pasos: botón derecho >Create >Procedural >Foliage.

- ***Prefab Properties***
  - **Prefab**: Objeto que se quiere instanciar
  - **Distance Separation**: Boleando que indica si el objeto que queremos instanciar tiene espacio libre a su alrededor, un ejemplo claro del uso del mismo es la hierba ya que nos da igual que haya hierba alrededor de hierba
  - **Unit Space Separation**: Unidades de Unity de separación que requiere ese objeto con respecto a otros ("con el YES")

- ***Density Features***

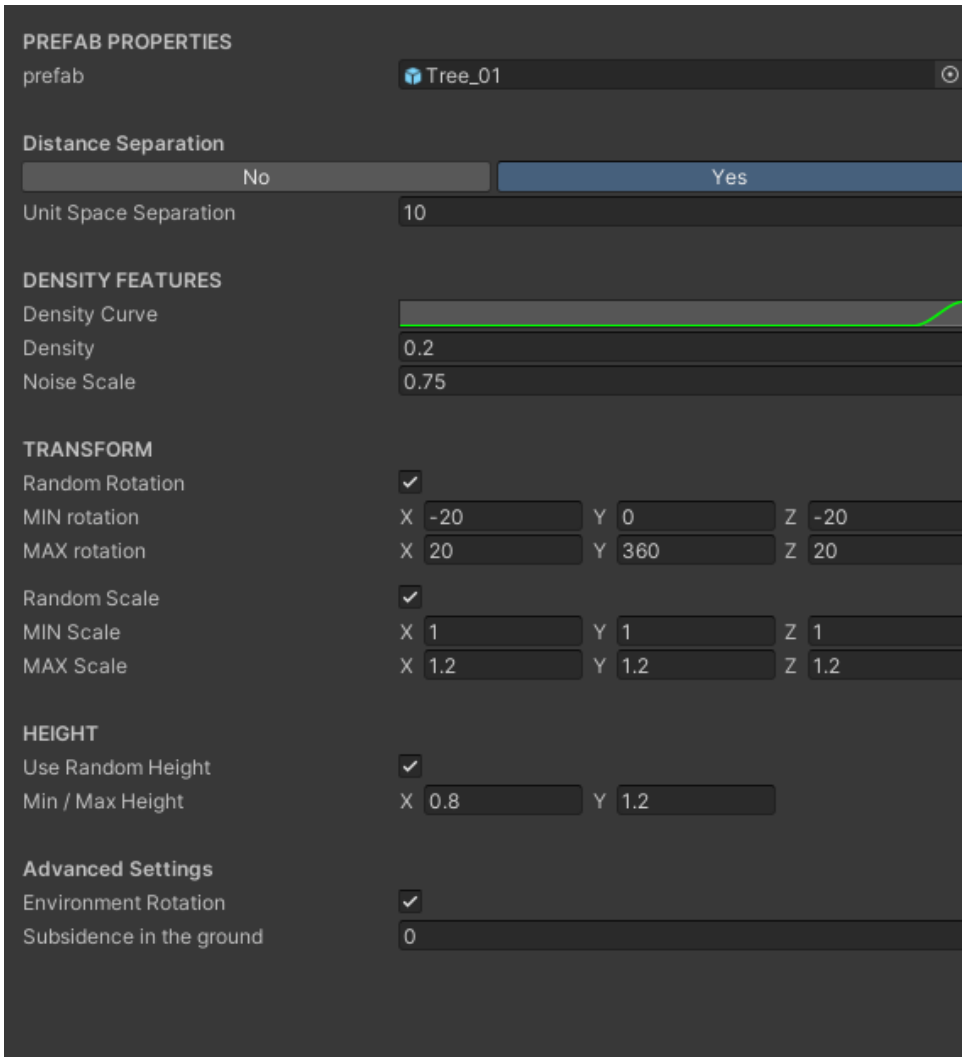


Figura D.3: Foliage Object

- **Density Curve**: Curva que permite establecer de qué parte a qué parte en términos de altura hay mas probabilidades de que se genere el objeto
- **Density**: Densidad del objeto que queremos instanciar

- ***Transform Properties***
  - **Rotation**
    - **Random Rotation**: Si el objeto va a tener rotación aleatoria
    - **Min Rotation**: Rotación mínima del objeto
    - **Max Rotation**: Rotación máxima del objeto
    - **Rotation**: Si el objeto no va a tener rotaciones aleatorias, qué rotación básica va a tener
  - **Scale**
    - **Random Scale**: Si el objeto va a tener escala aleatoria
    - **Min Scale**: Escala mínima del objeto
    - **Max Scale**: Escala máxima del objeto
    - **Scale**: Si el objeto no va a tener escala aleatoria, qué escala básica va a tener el objeto

D.1.3. TextureUpdater

La malla de terreno tiene asociado un material que utiliza nuestro shader personalizado Este componente se encarga de actualizar y modificar los parámetros de dicho material. El componente obtiene toda esta información de un ScriptableObject de tipo TextureData, dentro de este scriptableObject se puede configurar todos los parámetros referentes al apartado gráfico del terreno. De esta forma, el usuario puede tener al mismo tiempo distintos TextureData para definir distintas capas y texturas. Cabe resaltar que el terreno sólo utiliza un TextureData al mismo tiempo.

El scriptableObject "TextureData" permite definir hasta 8 capas en las que se puede dividir el terreno, cada una de estas capas tiene los siguientes parámetros:

- **Texture**: Textura que se va a utilizar en esta sección del terreno. Esta textura debe ser tileable y accesible y modificable, por lo que el sprite debe importarse con una configuración específica, asignando el parámetro ReadOnly a false. Cabe resaltar que no es obligatorio asignar una textura, la herramienta entonces tendrá en cuenta el color elegido
- **Tint**: La herramienta también te da la opción de tintar la capa, combinando el color seleccionado con la textura. También es una opción no asignar un color, en este caso no se modificará el color de la textura

- **TintStrenght**: Se constituye como un valor decimal entre 0 y 1, si el valor es cercano al 0, el color tendrá menos efecto sobre la textura mientras que si es cercano al 1, la textura se vera más afectada

- **StartHeight**: Este parámetro se encarga de definir la altura desde la que se empieza a renderizar esta capa. Constituye un valor decimal entre 0 y 1, siendo 0 el punto más bajo del terreno y 1 el punto más alto del terreno

- **Blend Strenght**: También constituye un valor entre 0 y 1. Define el suavizado de la textura de esta capa con el resto de capas. Si el valor es cercano al 0, el cambio entre las texturas de las capas es inmediato, sin ningún tipo de suavizado, mientras que si es cercano al 1, las demás capas se verán afectadas por esta de forma más notable

- **Texture Scale**: Define la escala con la que se renderizará la textura de esta capa

Una vez que el usuario modifica los parámetros que vea conveniente, se aplican presionando el botón de "Update" situado abajo del todo. También existe una alternativa que es el "AutoUpdate", que actualiza los valores internos del shader cada vez que se modifica cualquier parámetro, esto se usa para hacer pequeños cambios y actualizar el terreno de forma instantánea.

D.1.4. Endless Terrain

Endless terrain es un componente (Script) que permite mejorar la eficiencia de la escena con el mapa. A pesar de su nombre no genera un mapa infinito. Su función consiste en ocultar los chunks que se encuentran a una distancia configurable de la figura del jugador. El mapa que se puede ver ha sido generado previamente siguiendo la configuración del Map Generator. Pero no se verá hasta ejecutar Unity (véase en Figura D.5).

Instrucciones:

- Si el objeto map3D tiene algo dentro debe ser eliminado

- En el objeto Map Generator hay que activar el componente Endless Terrain

- Asegurarse de que el componente anterior tiene asignado un objeto que actúe de jugador

- Dar al play de Unity

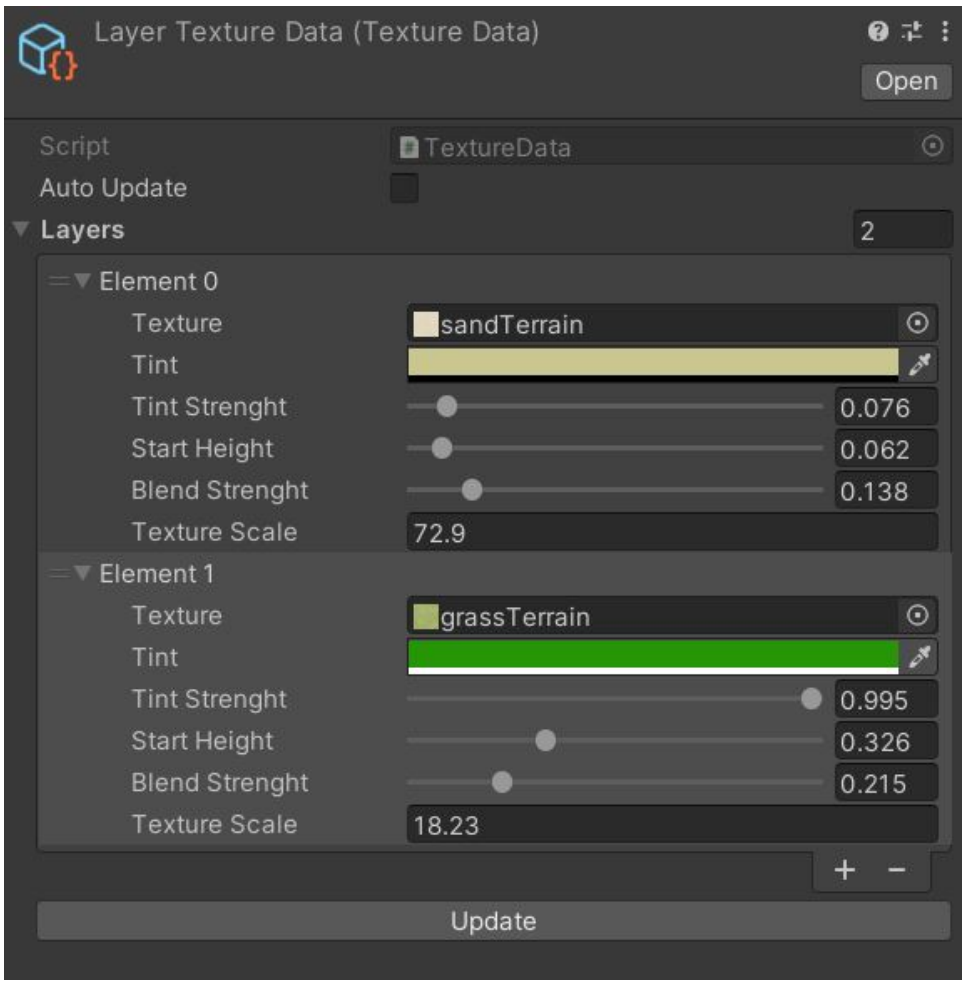
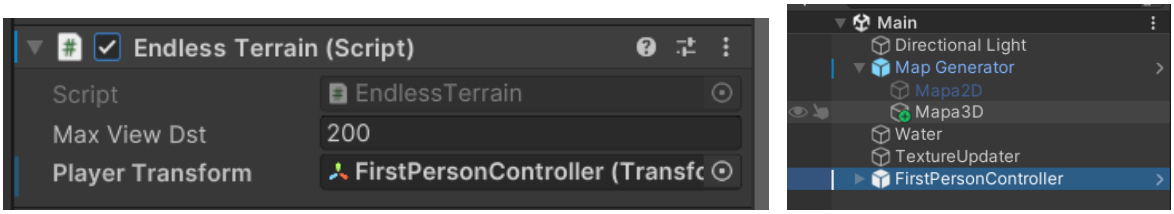


Figura D.4: Texture Updater Controller



(a) EndlessTerrainComponent (b) Jerarquía Editor

Figura D.5: EndLess Terrain Component

D.1.5. Interest Point

Este objeto (véase en Figura D.6) permite definir las características y parámetros de un punto de interés. Para crear un objeto de este tipo se deben seguir estos pasos: botón derecho >Create >Procedural >Interest Point.

- ***General***
  - **Type of Object**: Permite seleccionar un tipo dentro de una selección (Resources, Construction, Enemy, Mission, Allies) para hacer una diferenciación entre ellos
  - **Object Instance**: Prefab que representa a ese punto de interés (ej: para el castillo el prefab asociado es una torre)
  - **Amount**: Número de instancias que se van a generar de este objeto
  - **Attempts**: Número de intentos del algoritmo para encontrar una posición válida para este objeto
- ***Generate Settings***
  - **Radius**: Radio de distancia entre los puntos. Sinónimo de dispersión en este caso
  - **Max Height**: Altura máxima de generación
  - **Min Height**: Altura mínima de generación
  - **Biomes**: Lista de biomas en los que se genera

D.1.6. Map Display

Componente que permite dibujar o renderizar el mapa generado en 2D.

- **TextureRenderer**: Render del objeto en el cual se va a dibujar el mapa 2D

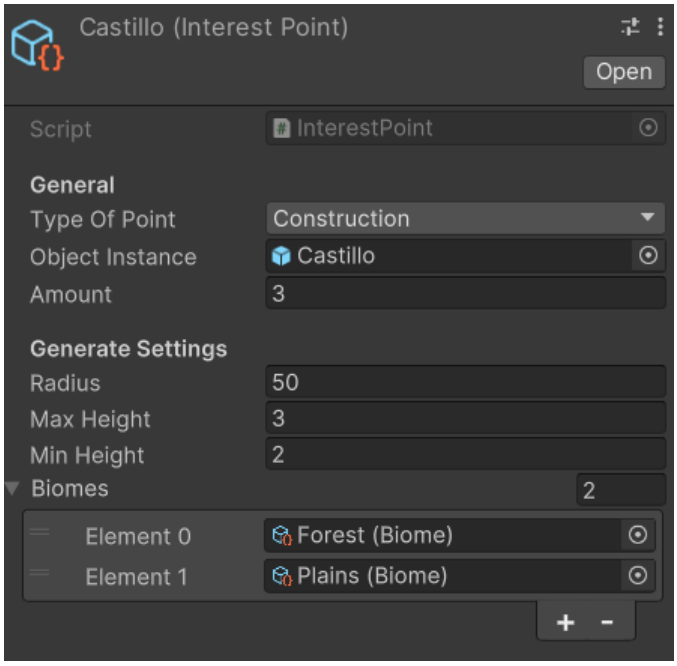


Figura D.6: Interest Point Object

D.1.7. WindShader

Este shader se encuentra en la carpeta Assets del proyecto. Permite dar un efecto de viento a los objetos a los que se aplica. Para poder aplicarlo es necesario seguir estos pasos:

1. Crear un nuevo material
2. En el apartado shader seleccionar: Shader Graphs/WindShader
3. En el apartado MainColor elegir el color base que se desea que tenga el objeto. Se suma a la textura que posea. Para no interferir en el color de esta, elegir color blanco.
4. En el apartado MainAlbedo colocar la textura
5. En el apartado WindMovement seleccionar en que eje se produce el movimiento. Su valor numérico indicara la frecuencia del movimiento
6. Apartado WindDensity para modificar el ruido
7. Apartado WindStrengh para modificar la fuerza del viento
8. Aplicar ese material al objeto deseado

Se puede ver un ejemplo en Figura D.7.

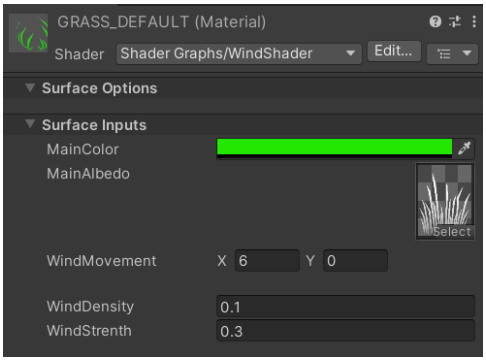


Figura D.7: Material con el shader de viento aplicado

D.2. Limitación de responsabilidades

No será responsable de ningún daño directo, indirecto, incidental, especial, consecuente o punitivo, incluyendo, pero no limitado a, pérdida de ingresos, pérdida de beneficios, pérdida de datos, interrupción del negocio o cualquier otro daño similar, derivado del uso o la imposibilidad de utilizar la herramienta proporcionada. El usuario reconoce y acepta que la herramienta se proporciona "tal cual" y que Procedural OVO Worlds no ofrece garantías de ningún tipo, ya sean expresas o implícitas, incluyendo, pero no limitado a, garantías de comerciabilidad, idoneidad para un propósito particular, o no infracción. Procedural OVO worlds tampoco será responsable de cualquier daño que pueda surgir como resultado de acciones realizadas por terceros. La responsabilidad total de Procedural OVO worlds, ya sea en contrato, agravio (incluyendo negligencia) o de otro modo.

Apéndice E

Entrevistas

En este apéndice se muestra el guión de la entrevista. A continuación aparecen las entrevistas a cada uno de los probadores. Se detallan tanto las preguntas como sus respuestas.

E.1. Guión

- ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?
- ¿Entiendes cómo funcionan todas las variables?
- ¿Cuáles son las menos intuitivas?
- ¿Consideras que el número de ajustes en los biomas es el adecuado?
- ¿Y en el follaje?
- ¿Y en el MapGenerator en sí?
- ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?
- ¿Qué te hubiera gustado modificar?
- ¿Utilizarías esta herramienta?
- ¿En qué contexto?
- ¿Qué funcionalidades añadirías a la herramienta?
- ¿Qué aspectos mejorarías?
- Cualquier pregunta que surja a partir de lo observado de las pruebas
- Comentarios adicionales

E.2. Entrevistas de los probadores

E.2.1. Probadora Paula

Q - ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?

A - Sin ayuda es difícil. Hace falta guía. Las curvas son difíciles de entender.

Q - ¿Entiendes cómo funcionan todas las variables?

A - No todas.

Q - ¿Cuáles son las menos intuitivas?

A - Alturas, curvas, transiciones y ruido.

Q - ¿Consideras que el número de ajustes en los biomas es el adecuado?

A - Si, bastante. Pero son abrumadores, hay muchos.

Q - ¿Y en el follaje?

A - Es mas intuitivo, quizás porque ya he trabajado con la herramienta de Unreal y es bastante parecido.

Q - ¿Y en el MapGenerator en sí?

A - Está bien que esté organizado por secciones. Me gusta que lo de los biomas y el foliage (vegetación) estén separados de lo que es el generador de mapas.

Q - ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?

A - Estoy la verdad que MUY contenta con el resultado. Se parece bastante.

Q - ¿Qué te hubiera gustado modificar?

A - (...)

Q - ¿Utilizarías esta herramienta?

A - Sí.

Q - ¿En qué contexto?

A - En entornos a gran escala. En proyectos con mapas muy grandes.

Q - ¿Qué funcionalidades añadirías a la herramienta?

A - Mas variedad de puntos de interés.

Q - ¿Qué aspectos mejorarías?

A - Explicar de alguna forma las cosas menos intuitivas como las curvas, ya que antes no me aparecían árboles y no sabía por qué era y resultó ser una curva. Hacer algún tutorial.

Comentarios adicionales

A la pregunta de funcionalidades que le gustaría añadir no respondió con una funcionalidad, así que no debemos tenerlo en cuenta.

La probadora necesitó que la prueba fuera bastante guiada. Fueron necesarias muchas pistas.

E.2.2. Probador Víctor

Q - ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?

A - De primeras me resulto complicado, pero a medida que iba trabajando con la herramienta iba entendiendo como funcionaba mas o menos todo y fue resultando mas sencillo

Q - ¿Entiendes cómo funcionan todas las variables?

A - Todas no, pero bastantes sí, estoy seguro de que si hubiese tenido mas tiempo hubiese podido acordarme de mas.

Q - ¿Cuáles son las menos intuitivas?

A - Las Transiciones de biomas, porque no entendía muy bien como afectaba la curva a la generación del terreno. Además no entendía muy bien las variables de los biomas de Noise Settings y me liaba en el foliage (vegetación) entre height y scale pensaba que height era la altura en la que aparecería el objeto.

Q - ¿Consideras que el número de ajustes en los biomas es el adecuado?

A - En general todo bien, no he echado nada en falta.

Q - ¿Y en el follaje?

A - También todo bien excepto lo que he dicho anteriormente con el height y scale

Q - ¿Y en el MapGenerator en sí?

A - Todo bien

Q - ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?

A - Si

Q - ¿Qué te hubiera gustado modificar?

A - Que los puntos de interés estén asociados al bioma y no a todo el mapa, además que estos se puedan bloquear de alguna manera para que si quiero generar de nuevo todo el mapa y con ello los puntos de interés solo se regeneren los que no me han gustado.

Q - ¿Utilizarías esta herramienta?

A - La usaría en el 60 % del mapa que quiero hacer, me solucionaría el problema de papel en blanco a la hora de empezar un proyecto de un juego de mundo abierto con el mapa, ya que no sabría que quiero tener de primeras.

Q - ¿En qué contexto?

A - GameJams y juegos propios

Q - ¿Qué funcionalidades añadirías a la herramienta?

A - Las dichas anteriormente de los puntos de interés. Poder solo volver a generar aquellas zonas que no me han gustado como han quedado y poder quedarme con esas otras que han gustado.

Q - ¿Qué aspectos mejorarías?

A - Los dichos anteriormente.

Comentarios adicionales:
General todo bien, le costaba mucho al principio entrar en la dinámica de trabajo y preguntaba al principio que hacían casi todas las variables que usaba.

E.2.3. Probador Rodrigo

Q - ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?

A - De primeras cuesta un poco, pero luego a medida que pruebas un poco se vuelve más fácil.

Q - ¿Entiendes cómo funcionan todas las variables?

A - Todas no.

Q - ¿Cuáles son las menos intuitivas?

A - Las transiciones de biomas. No entiendo muy bien los parámetros del noise y qué hace cada uno exactamente.

Q - ¿Consideras que el número de ajustes en los biomas es el adecuado?

A - En general todo bien.

Q - ¿Y en el follaje?

A - Bastante bien en general. Se entiende todo, aunque el nombre del parámetro de la distancia de separación es demasiado largo, hay demasiado texto.

Q - ¿Y en el MapGenerator en sí?

A - Me parece que esta todo bien.

Q - ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?

A - Sí.

Q - ¿Qué te hubiera gustado modificar?

A - Me hubiera gustado poder decir donde quiero que se genere un bioma en concreto.

Q - ¿Utilizarías esta herramienta?

A -Si, para pintar un mapa de fondo o un mapa desde donde poder empezar a definir que quiero en mi videojuego.

Q - ¿En qué contexto?

A - En GameJams y juegos que hiciera yo.

Q - ¿Qué funcionalidades añadirías a la herramienta?

A - Poder pintar un mapa de biomas. Yo pinto el mapa y luego se genera.

Q - ¿Qué aspectos mejorarías?

A - Lo que he dicho antes, pintar los mapas y poder seleccionar dónde quiero un determinado bioma.

Comentarios adicionales:
En general todo bien, le costaba mucho al principio entrar en la dinámica de trabajo.

E.2.4. Probadora Elisa

Q - ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?

A - EL proceso de aprendizaje es intuitivo y al tener los elementos separados por funcionalidad, ayuda a encontrar mas fácilmente la localización de un parámetro en concreto.

Q - ¿Entiendes cómo funcionan todas las variables?

A - Existen algunos parámetros que necesitaría una explicación mas detallada para entender su función, como las curvas.

Q - ¿Cuáles son las menos intuitivas?

A - Las curvas que definen la densidad de un objeto por altura es difícil de entender de primeras, aunque después te acostumbras.

Q - ¿Consideras que el número de ajustes en los biomas es el adecuado?

A - Si, no llega a agobiar el numero de opciones.

Q - ¿Y en el follaje?

A - (No aplica, no llegó a modificar ningún archivo de follaje).

Q - ¿Y en el MapGenerator en sí?

A - Si.

Q - ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?

A - Si.

Q - ¿Qué te hubiera gustado modificar?

A - Nada.

Q - ¿Utilizarías esta herramienta?

A - Sí.

Q - ¿En qué contexto?

A - En demos de programación, para tener un fondo bonito y personalizado, sin la necesidad de invertir mucho tiempo.

Q - ¿Qué funcionalidades añadirías a la herramienta?¿Qué as-

pectos mejorarías?

A - Sobretodo añadiría documentación rápidamente accesible.

E.2.5. Probador Miguel

Q - ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?

A - Las funciones de la herramienta se entienden rápido y son intuitivas de usar.

Q - ¿Entiendes cómo funcionan todas las variables?

A - Las que tienen que ver con los detalles de la generación de las mallas no, como las octavas.

Q - ¿Cuáles son las menos intuitivas?

A - Las curvas que definen las transiciones.

Q - ¿Consideras que el número de ajustes en los biomas es el adecuado?

A - Que tampoco haya muchísimos parámetros editables, hace que la herramienta sea mas sencilla y agradable de aprender.

Q - ¿Y en el follaje?

A - (No aplica, no llegó a modificar ningún archivo de follaje).

Q - ¿Y en el MapGenerator en sí?

A - Tiene un numero de parámetros aceptables, sin llegar a sobrecargar de información la herramienta.

Q - ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?

A - Si.

Q - ¿Qué te hubiera gustado modificar?

A - Nada.

Q - ¿Utilizarías esta herramienta?

A - Sí.

Q - ¿En qué contexto?

A - Lo utilizaría para crear escenarios de forma rápida en GameJams o para demos de programación.

Q - ¿Qué funcionalidades añadirías a la herramienta?¿Qué aspectos mejorarías?

A - Una funcionalidad Toolkit en la que al pasar el ratón por un parámetro que no entiendas, salga un pequeño resumen explicando que efectos tiene ese parámetro sobre el resultado final.

E.2.6. Probadora Rocío

Q - ¿Cómo de difícil consideras que es acostumbrarse al flujo de trabajo de la herramienta?

A - No es muy difícil, en plan solo tienes dos componentes para mirar principalmente, el grande y los biomas.

Q - ¿Entiendes cómo funcionan todas las variables?

A - Todas no.

Q - ¿Cuáles son las menos intuitivas?

A - Las curvas sigo sin saber exactamente cómo funcionan.

Q - ¿Consideras que el número de ajustes en los biomas es el adecuado?

A - Como algo genérico está bastante bien, lo del offset te ayuda a ver la sección exacta que quieres.

Q - ¿Y en el follaje?

A - (No aplica, no llegó a modificar ningún archivo de follaje).

Q - ¿Y en el MapGenerator en sí?

A - Opino igual que con los biomas, para algo así de genérico esta muy bien.

Q - ¿Conseguiste crear un terreno parecido a las imágenes/acorde con cómo te lo imaginabas/Estás contento/a con los terrenos resultantes?

A - Si.

Q - ¿Qué te hubiera gustado modificar?

A - Nada.

Q - ¿Utilizarías esta herramienta?

A - Sí.

Q - ¿En qué contexto?

A - Para rollo juegos que necesiten cosas generadas de manera procedural, como un roguelike.

Q - ¿Qué funcionalidades añadirías a la herramienta?¿Qué aspectos mejorarías?

A - Hay muy poco control sobre qué biomas salen juntos, ya que es imposible hacer que un bioma pueda aparecer al lado de otros tres biomas.

Comentarios adicionales:

Consiguió recrear la imagen bastante rápido, de primeras no se le ocurrió utilizar la función del follaje para dejarlo perfecto.

Apéndice F

Algoritmos

En este capítulo se muestra el diseño de distintos algoritmos usados en el proyecto mediante pseudocódigo.

F.1. Algoritmo para la generación de los mapas de ruido

```
1: class: Noise
2: function GenerateNoiseMap(int size,int seed, NoiseSettings noiseSettings)
   ->[,]
3: if ( thennoiseSettings.noiseScale <= 0)
4:     noiseSettings.noiseScale = 0.0001f
5: end if
6: [,] noiseMap = new float[size, size]
7: Random r = new Random(seed)
8: octaveOffsets = Vector2[noiseSettings.octaves]
9: for ( doint i = 0; i <noiseSettings.octaves; i++)
10:     offsetX = r.Next(-10000, 10000) + noiseSettings.offset.x
11:     offsetY = r.Next(-10000, 10000) + noiseSettings.offset.y;
12:     octaveOffsets[i] = Vector2(offsetX, offsetY);
13: end for
14: maxNoiseHeight = float.MinValue;
15: minNoiseHeight = float.MaxValue;
16: halfSize = size / 2f;
17: for ( doint y = 0; y <size; y++)
18:     for ( doint x = 0; x <size; x++)
19:         amplitude = 1
20:         frequency = 1
21:         noiseHeight = 0
22:         for ( doint i = 0; i <noiseSettings.octaves; i++)
23:             smpleX = (x - halfSize) / noiseSettings.noiseScale * frequency
```

```
    + octaveOffsets[i].x
24:             smpleY = (y - halfSize) / noiseSettings.noiseScale * frequency
    + octaveOffsets[i].y
25:             perlinValue = PerlinNoise(smpleX, smpleY) * 2 - 1
26:             noiseHeight += perlinValue * amplitude
27:             amplitude *= noiseSettings.persistance
28:             frequency *= noiseSettings.lacunarity
29:         end for
30:         if ( thennoiseHeight >maxNoiseHeight)
31:             maxNoiseHeight = noiseHeight
32:         end if(niseHeight <minNoiseHeight) minNoiseHeight = noiseHeight
33:         noiseMap[x, y] = noiseHeight
34:     end for
35: end for
36: for ( doint y = 0; y <size; y++)
37:     for ( doint x = 0; x <size; x++)
38:         noiseMap[x, y] = Mathf.InverseLerp(minNoiseHeight, maxNoiseHeight,
    noiseMap[x, y])
39:     end for
40: end for
41: return noiseMap
```

F.2. Algoritmo para la generación de puntos de interés

```
    class: PoissonDiscSampler
2:      spawnPoints: List<Vector2>
        points: List<Vector2>
4:      grid: 2D[]
        attempts: int
6:      radius: float
        amount: int
8:      cellSize: float

10: PoissonDiscSampler (s:float, rfloat, amount: int, att: int, maxH: int,
    minH: int, mapInfo_: MapInfo, mapGen_: MapGenerator, biomes_:
    Biome[])
        spawnPoints = spawnPoints
12:     points = []
        grid = int[s / cellSize, (s / cellSize]
14:     attempts = attempts
        radius = radius
16:     amount = amount
```

```
    cellSize = cellSize
18:
function generate() ->List<Vector2>
20: while spawnPoints.Count > 0 do
    spawnIndex = randomRange(0, size(spawnPoints) - 1)
22:  spawnCentre = spawnPoints[spawnIndex]
    candidateAccepted = false
24:  for i ← 0 to attempts - 1 do
        angle = randomValue() * 2 * π
26:     dir = Vector2(sin(angle), cos(angle))
        candidate = spawnCentre + dir * randomRange(radius, 2 * ra-
dius)
28:     if isValid(candidate) and points.Count < amount then
            points += candidate
30:         spawnPoints += candidate
            grid[int(candidate.x / cellSize)][int(candidate.y / cellSize)] =
points.Count
32:         candidateAccepted = true
            break
34:     end if
    end for
36: if not candidateAccepted then
        spawnPoints -= spawnIndex
38: end if
end while
40:
return points
```

F.3. Algoritmo para la generación de la localización de biomas

```
class: BiomeGenerator
function GenerateBiomes(int seed,int size, offset)
3: if length(biomes) >255 then
    show error "Maximum number of biomes allowed is 255, please con-
sider reducing the amount, truncating array"
    copy(biomes, biomes, 255)
6: end if
configure noise settings:
  noiseScale = noiseSize
9:  octaves = 3
    lacunarity = 1
    offset = offset
```

```
    persistence = 1
biomesNoise = GENERATENOISEMAP(size, seed + 1, noiseSettings) (as described in Algorithm 0)
biomeMap = byte matrix of size [size, size]
influences = dictionary of size [size, size]
defaultThreshold = 1 / length(biomes)
thresholds = float array of length(biomes)
thresholds[0] = biomes[0].density * defaultThreshold
for i from 1 to length(biomes) - 1 do
    thresholds[i] = thresholds[i - 1] + biomes[i].density * defaultThreshold
end for
for i from 0 to size - 1 do
    for j from 0 to size - 1 do
        noise = biomesNoise[i, j]
        noiseValue = noise * thresholds[last index]
        currentBiomeIndex = 0
        biomeInfluence = 0
        for x from 0 to length(biomes) - 1 do
            if noiseValue ≤ thresholds[x] then
                lowerValue = 0
                if x >0 then
                    lowerValue = thresholds[x - 1]
                else
                    influences[i, j] = new dictionary
                    add to dictionary influences[i, j] biomes[x] with value 1
                    break
                end if
                upperValue = thresholds[x]
                biomeInfluence = (noiseValue - lowerValue) / (upperValue - lowerValue)
                firstBiomeInfluence = evaluateTransitionCurve(biomeInfluence)
                secondBiomeInfluence = evaluateTransitionCurve(1 - biomeInfluence)
                influenceMagnitude = firstBiomeInfluence + secondBiomeInfluence
                influences[i, j] = new dictionary
                add to dictionary influences[i, j] biomes[x] with value firstBiomeInfluence / influenceMagnitude
                if biomes[x] == biomes[x - 1] then
                    increase value in dictionary influences[i, j] biomes[x - 1] with value secondBiomeInfluence / influenceMagnitude
                else
                    add to dictionary influences[i, j] biomes[x - 1] with value secondBiomeInfluence / influenceMagnitude
```

```
                end if
                currentBiomeIndex = if biomeInfluence >0.5 then x else
    x - 1
51:             break
            end if
        end for
54:     biomeMap[i, j] = currentBiomeIndex
    end for
end for
```

Bibliografía

Archer, T. Procedurally generating terrain. En *44th annual midwest instruction and computing symposium, Duluth*, páginas 378–393. 2011.

Diego, R. S. Red dead redemption. 2010.

Digital, P. Gran turismo 5. 2010.

docs.unity3d.com. - anatomía de una malla (mesh). 2021.

Entertainment, N. S. Roboblitz. 2006.

de Empresas Desarrolladoras de Videojuegos y Software de Entretenimiento), D. A. E. *Libro Blanco del Desarrollo Español de Videojuegos 2022*. DEV, Madrid, España, 2022.

Epic Games. Pcg development guides. 2023.

formacionprofesional.net. - ¿qué son los shaders en el mundo de la animación 3d, juegos y entornos interactivos. ????

Games, H. No man's sky. 2016.

Generation, P. C. Simplex noise. 2024.

LEGO Group. The adventure is building: Lego fortnite is live. 2023.

Maxis. Spore. 2008.

Millington, I. y Funge, J. *Artificial Intelligence for Games, Second Edition*. 2009. ISBN 9780123747310.

Nintendo EPD. The legend of zelda: Breath of the wild. Nintendo Switch, Wii U, 2017. Video game.

Pahunov, D. Mapmagic world generator. 2023.

Rockstar North. Grand theft auto v. PlayStation 3, Xbox 360, PlayStation 4, Xbox One, Microsoft Windows, PlayStation 5, Xbox Series X/S, 2013. Video game.

Rockstar Studios. Red dead redemption 2. PlayStation 4, Xbox One, Microsoft Windows, Stadia, 2018. Video game.

Sell, S. Gpu accelerated diamond-square generation. 2024.

SideFX. Houdini indie. 2023.

Software, G. Borderlands. 2009.

Technologies, U. Unity: Real-time development platform. 2024.

Ubisoft. Tom clancy's the division. 2016.

Unity Asset Store. Gaia pro 2021 - terrain & scene generator. 2021.

Unity Asset Store. Houdini engine for unity. 2023a.

Unity Asset Store. Terraforge beta - procedural terrain generator. 2023b.

Unity Asset Store. Terraworld 2023 - node based real-world 3d terrain tool. 2023c.

Unity Asset Store. World creator standard. 2023d.

y Zach Adams, T. Dwarf fortress. 2006.