Generación aleatoria de terrenos 3D con Unity

Random 3D Terrain Generation with Unity

Airam González Hernández

Escuela Superior de Informática y Tecnologías
Sección de Ingeniería Informática

La Laguna, 08 de Septiembre de 2015

D. **Carina Soledad González González**, con N.I.F. 54064251-Z profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“Generación aleatoria de terrenos 3D con Unity."

ha sido realizada bajo su dirección por D. **Airam González Hernández**,
con N.I.F. 42195682-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 08 de Septiembre de 2015.

Agradecimientos

A Carina:

Porque gracias a ella he podido realizar este TFG en una temática que me ha gustado muchísimo, por haberme aconsejado sobre cómo ser mejor profesional y haberme dado la posibilidad de ser totalmente libre, creativo y permitiendo tomar las decisiones sobre cómo orientar su desarrollo y siempre ofreciendo críticas constructivas al respecto sin ponerme ningún tipo de limitaciones.

A Alberto Erice:

Por haberme guiado a lo largo de Unity, intentando a su vez convertirme en autosuficiente y autodidacta. Por haberme brindado la oportunidad de entrar a este mundo y mostrarme como es el día a día en una empresa orientada al desarrollo de videojuegos.

A mis padres:

Porque sin ellos nada de esto habría sido posible, nunca hubiese podido ser ni técnico ni ingeniero. Por haberme apoyado incondicionalmente y demostrar una confianza ciega en todo momento. Por nunca dudar de mis posibilidades y siempre animarme a seguir luchando por mis metas.

A Carmen:

Por esperar siempre cosas grandes e impresionantes de mi. Por hacerme creer en todo momento que soy bueno en lo que hago y no dudar de mi capacidad de alcanzar una meta. Por haberme apoyado todos los días en la realización de este proyecto y convencerme de mi capacidad para desarrollarlo.

A mis amigos:

Por mentir descaradamente cada vez que les preguntaba como veian mi progresión en el proyecto y siempre decirme que era impresionantemente bueno.

Al profesorado de la antigua ETSII (nueva ESIT):

Porque cada uno de los docentes presentes a lo largo de mi carrera ha dejado su pequeño recuerdo en mi. Todos y cada uno de ellos ha contribuido a que haya llegado hasta aquí.

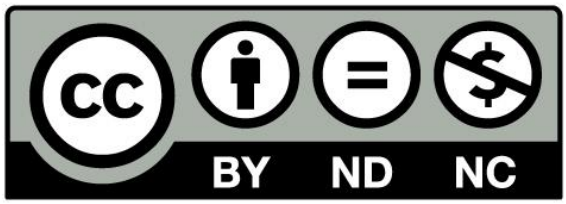
A Julian Vera:

Por su ayuda desinteresada en lo relacionado al mundo del desarrollo de videojuegos y su interés en mi progreso.

A darkstar:

Por su gran guía a la hora de desarrollar el Perlin Noise y toda la parte relacionada con el ruido pseudoaleatorio.

Licencia



Resumen

El objetivo de este TFG ha sido realizar un generador de terrenos aleatorios en 3D con Unity [(1)] [(2)] *en conjunto con la empresa Adamantite Software SL.*

Generar un terreno en Unity es una labor que requiere bastante tiempo y dedicación. Automatizar esta tarea supondría un ahorro de recursos bastante importante para las empresas desarrolladoras (tiempo / empleados).

Previo al comienzo de este TFG no existían alternativas automatizadas para la realización de esta tarea en esta tecnología (Unity). Si se deseaba construir un terreno la única forma era mediante el asistente del framework de Unity, o que cada usuario realizase su propio código para su terreno desde cero.

Los principales problemas a la hora de generar un terreno, eran modelarlo, texturizarlo (aplicar texturas que le den colores naturales) y añadirle objetos que lo hagan parecer realista. Todas estas fases consumen muchísimo tiempo de desarrollo, ya que, pese a la potencia del framework de Unity, realizar esta tarea a mano es una tarea muy extensa.

Para solucionar esta problemática, se ha desarrollado el generador aleatorio, que consta de tres tareas principales. Generar las formas del terreno natural mediante un algoritmo de ruido aleatorio. Texturizar este terreno con las texturas deseadas y aplicar árboles y detalles de manera aleatoria por el terreno ya generado. Todo esto concentrado en una interfaz sencilla que se basa en muy pocos parámetros, a deseo del usuario.

Palabras clave: Unity, videojuego, generador, terreno.

Abstract

The main goal of this project has being develop a random 3D terrain generator with Unity with the collaboration of Adamantite Software SL.

Build a terrain with Unity is a task which requires a lot of time and dedication. Automatice this task will be a big resource saving for the developers (saving time and employes).

At the beginning of this project there was no choices which could automatice this task with this technology (Unity). If you wanted to build a terrain, the only way to do it was by Unity framework, or the users also develop their own code for their terrain from the start.

The main problems while building terrains, were model it, texturize it (Apply textures in order to "naturalize") and add some objects which would make it appears real. Al this phases cost tons of developing time, because, even with the Framework potency, doing this tasks by your own hand is a very extensive work.

In order to solve this problems, a random generator has being developed, which is divided into three main tasks. Model a natural terrain with a random noise algorithm. Texturize this terrain with textures of your own and apply trees and details in a random way. All of this tasks comes in to an easy user interface based on parameters.

Keywords: Unity, videogame, generator, terrain.

Índice General

Agradecimientos **3**
Licencia **4**
Resumen **5**
Abstract **6**
Lista de Figuras **9**
Capítulo 1 ..... **10**
Introducción 10
1.1 Introducción a Unity. 10
1.2 Justificación y Objetivos. 11
1.3 Tecnologías utilizadas. 11
1.4 Organización de la memoria. 12
Capítulo 2 ..... **13**
Definiciones 13
2.1 GameObject. 13
2.2 Alpha Blending. 13
2.3 Billboarding. 13
Capítulo 3 ..... **15**
Generación de un terreno mediante algoritmos de ruido 15
3.1 Selección del algoritmo de ruido. 15
3.2 Funcionamiento del Perlin Noise. 15
3.3 Aplicación del Perlin Noise a la clase terrain. 21
Capítulo 4 ..... **24**
Texturización del terreno 24
4.1 Métodos de texturización. 24
4.2 Texturización mediante código. 25
Capítulo 5 ..... **28**
Generación de vegetación 28
5.1 Generación y distribución de árboles. 28
5.2 Detallado del mapa. Hierba. 30
Capítulo 6 ..... **32**
Interfaz de usuario 32
Capítulo 7 ..... **33**
Conclusiones 33

Capítulo 8 34
Conclusions 34
Capítulo 9 35
Líneas de futuro 35
Anexos 36
Anexo 1: Código editTerrain 36
Anexo 2: Altura máxima 36
Anexo 3: Alpha Cortes 36
Anexo 4: Dibujado de árboles 37
Anexo 5: Dibujado de hierba 38
Bibliografía 39

Lista de Figuras

Figura 1: Editor de terrenos 10
Figura 2: f(x) = Sen(x) 16
Figura 3: f(x)=Sen(x*2)/1.5 16
Figura 4: f(x)=Sen(x*4)/2 16
Figura 5: f(x)=Sen(x*20)/10 17
Figura 6: f(x)=sen(x) + sen(x*2)/1.5 + sen(x*4)/2 + sen(x*20)/10 17
Figura 7: Ejemplo de nodos sin interpolación 18
Figura 8: Primera Octava 18
Figura 9: Segunda Octava 18
Figura 10: Tercera octava 19
Figura 11: Cuarta octava 19
Figura 12: Quinta octava 19
Figura 13: Sexta octava 20
Figura 14: Séptima octava 20
Figura 15: Ruido Perlin para el terreno. 20
Figura 16: Creación de un terrain mediante editor. 21
Figura 17: Generación de terreno utilizando. 22
Figura 18: Terreno generado aleatoriamente. 23
Figura 19: Pintado de texturas con el editor. 24
Figura 20: Texturizado por altura SIN alpha blending. 26
Figura 21: Texturizado por altura CON alpha blending. 4 Texturas. 26
Figura 22: Texturizado por altura CON alpha blending. 4 Texturas. 27
Figura 23: Pintado de árboles con el editor. 28
Figura 24: Arboleda al 30% de altura. 29
Figura 25: Arboleda al 90% de altura. Diferente semilla. 30
Figura 26: Detalles en el terreno. No son visibles al alejar un poco más la cámara. 31
Figura 27: Interfaz de Usuario. 32

Capítulo 1

Introducción

Unity 3D es un motor de videojuegos disponible para plataformas de Microsoft y Mac osx. Es un software muy popular ya que permite diseñar videojuegos para casi cualquier plataforma existente. Videoconsolas, windows, linux, mac, android, incluso, gracias a plugins web, juegos para navegador.

Este potente motor proporciona todo tipo de herramientas para facilitar la tarea de desarrollo de videojuegos, así como muchos asistentes que ayudan a que cualquier usuario del software pueda usarlo sin tener avanzados conocimientos en programación.

1.1 Introducción a Unity.

Unity proporciona potentes herramientas para el desarrollo de videojuegos mediante su editor.

Diferentes tipos de gameobjects predefinidos en el sistema, tales como cubos, esferas, cámaras, luces, terrenos, árboles, zonas de viento etc.

Entre estos, el interés principal de este TFG ha sido centrarse sobre el editor de terrenos.

Unity 3D proporciona un potente editor de terrenos (objetos de la clase terrain) que permite todo tipo de detalles. El principal problema es que funciona como una herramienta de diseño.

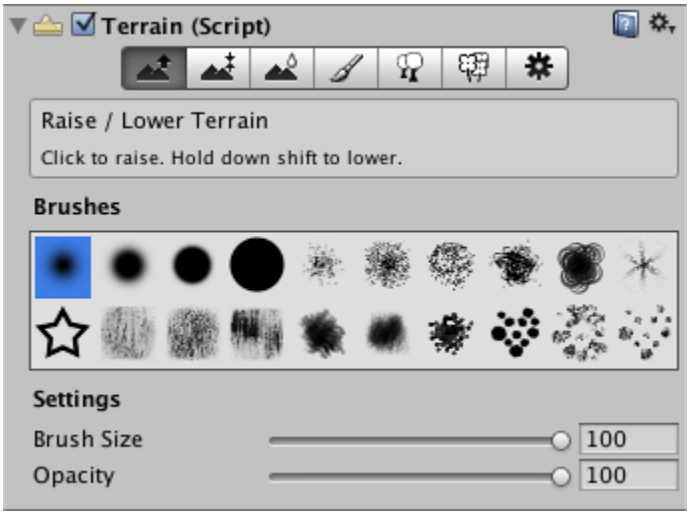


Figura 1: Editor de terrenos

Este editor nos permite modificar la topografía mediante un sistema de brochas y la herramienta de alturas. Este mismo editor proporciona una herramienta de texturizado, árboles, hierba y detalles así como zonas de viento.

1.2 Justificación y Objetivos.

El problema principal de Unity a la hora de editar terrenos, es que, salvo las zonas de viento, el resto de herramientas funcionan con el sistema de brochas. Por tanto, es un sistema bastante lento y tedioso.

Muchos desarrolladores, para evitar este método, generan mediante código sus propios terrenos, pero ninguno ha decidido construirlo como herramienta.

Este TFG ha tenido por objetivo el desarrollo de un plug-in/add-on para Unity que permite la generación aleatoria de terrenos 3D.

En el desarrollo del plugin, se han afrontado los sistemas de brocha, intentando automatizarlos por código, ofreciendo una interfaz sencilla de uso similar a la de Unity, pero evitando el sistema de brocha y añadiendo aleatoriedad a los terrenos generados.

De esta manera, los objetivos del TFG se pueden dividir en los siguientes:

- Generar una malla de terrenos de manera aleatoria con forma realista de manera automática.
- Aplicar texturas al terreno generado. Estas texturas deben aplicarse siguiendo un patrón lógico (en función de la altura, por ejemplo, arena en el más bajo, hierba en intermedios y nieve en altos).
- Añadir vegetación al terreno generado. Árboles y pequeñas flores o hierba.
- Construir una interfaz que facilite el uso de este generador a cualquier usuario sin necesidad de programación.

1.3 Tecnologías utilizadas.

Para el desarrollo de este TFG se han utilizado las siguientes tecnologías:

- Unity 3D: Como se ha mencionado en el apartado 1.1 Unity es un potente motor de desarrollo de videojuegos. Dentro de Unity nos hemos centrado en crear un script utilizando la clase Terrain [(3)].
  Esta clase provee los atributos necesarios para modificar cualquier cosa del terreno 3D. Ya sea su forma, sus contenidos, físicas etc.
- Visual Studio 2013 [(4)]: Entorno de desarrollo de microsoft, conjunto al plugin para Unity, que permite compilar las soluciones directamente y ejecutarlas al mismo tiempo en Unity.
- Lenguaje de programación C# [(5)]. El cual, junto con Javascript [(6)], son los dos lenguajes más populares en el desarrollo de Unity.

1.4 Organización de la memoria.

Esta memoria de TFG se organiza entonces de la siguiente manera:

- Capítulo 1: Introducción.
  Este capítulo trata una introducción al TFG, objetivo del mismo y organización de esta memoria.
- Capítulo 2: Definiciones
  En este capítulo introduciremos las definiciones básicas de los objetos Unity y atributos utilizados en el TFG
- Capítulo 3: Generación de un terreno mediante algoritmos de ruido.
  En este capítulo se tratarán los algoritmos de ruido estudiados y su aplicación en el plugin.
- Capítulo 4: Texturización del terreno.
  En este capítulo se tratará el tema de aplicación de texturas al terreno en función de su altura, así como la transición de una textura a otra sin efecto corte.
- Capítulo 5: Generación de vegetación.
  En este capítulo se tratará el tema de generación de vegetación para naturalizar más aún el terreno generado
- Capítulo 6: Interfaz de usuario.
  En este capítulo se tratará el tema de la interfaz de usuario.
- Capítulo 7: Conclusiones
- Capítulo 8: Conclusions.
- Capítulo 9: Líneas futuras.
- Bibliografía

Capítulo 2

Definiciones

2.1 GameObject.

Unity trabaja con un tipo de objetos particular, al que denomina GameObject [(7)].

Estos GameObject no son otra cosa que contenedores vacíos, pero que pueden contener cualquier componente del motor.

Por ejemplo, un GameObject del tipo Cube, tendrá por componentes un Transform (posición del objeto en nuestro espacio), un Mesh (malla del objeto, su forma), un Mesh renderer (renderizador de las texturas de la malla) y un Box Collider (El componente de colisiones, para dotar de física al objeto).

Para el plugin realizado, solo se crea un GameObject al que se le aplicará nuestro script de generador aleatorio. Este GameObject estará vacío, teniendo simplemente un Transform (que será la posición donde comienza a generarse el terreno) y nuestro script como segundo componente.

2.2 Alpha Blending.

El alpha blending [(8)] es una técnica de tratamiento de imágenes (texturas en nuestro caso) que nos permite jugar con el canal alpha de la imagen (textura).

Esta técnica hace que la transparencia de una textura cambie de acuerdo a unas condiciones dadas. En el caso de este plugin el Alpha Blending se necesita para facilitar la transición de una textura a otra sobre el terreno. Sin alpha blending una textura se sobrepondra a la otra, y, por tanto, se aprecia un salto o corte al cambiar de una textura a otra en el terreno.

Con Alpha Blending, se realiza una transición aumentando la transparencia de una textura al mismo tiempo que se disminuye la de la otra, de manera que toda la transición sea de manera coherente y continuada, pareciendo así natural.

2.3 Billboarding.

El efecto de Billboarding consiste en que una textura siempre esté mirando a la cámara activa en ese momento, de manera que ofrece un efecto de tridimensionalidad.

Realmente a nuestra textura (polígono u objeto, realmente se aplica a cualquier objeto que se desee) cuenta con un vector de orientación (un Transform). A medida que la posición de la cámara va cambiando, el transform de nuestro Billboard también lo hace.

Unity incorpora en su motor una gestión del Billboard de los objetos. Concretamente, los métodos de la clase TerrainData [(9)], tanto la gestión de árboles como la de detalles incluyen Billboard por defecto, y no se necesita añadirlo como componente.

Capítulo 3

Generación de un terreno mediante algoritmos de ruido

Los algoritmos de ruido son de amplia utilización en la informática multimedia (imágenes, sonidos y video) para generar fenómenos aleatorios (o pseudoaleatorios) Los algoritmos se encargan de generar ondas a partir de una entrada, de manera que para cada entrada conseguimos una onda diferente (lo que consideramos "ruido") Estas ondas pueden ser n-dimensionales, de manera que una onda bidimensional, por ejemplo, se utilizaría para generar una textura plana. Los valores de esta onda se traducen a escala de grises, de manera que la textura resultante pueda ser utilizada como heightmap, usadas como base para animaciones (por ejemplo, fuego) etc.

3.1 Selección del algoritmo de ruido.

Existen varios tipos de algoritmo de ruido, el algoritmo más clásico es el Perlin Noise [(10)].

También se ha valorado el Simplex Noise. Este ruido es menos complejo computacionalmente y permite escalar a mayores dimensiones (4 y 5 dimensiones) así como también es más sencillo para ser implementado en hardware.

Existen variaciones de estos algoritmos, como puede ser el Pink Noise, pero son más orientadas a otros tipos de ruido (como puede ser el ruido blanco)

En el caso de este TFG, se ha optado por realizar una implementación del Perlin Noise, ya que pese a tener mayor complejidad computacional, existía más información sobre implementaciones del mismo para generación de terrenos.

3.2 Funcionamiento del Perlin Noise.

Perlin Noise es un tipo de ruido basado en gradientes, desarrollado por Ken Perlin [(10)] en 1983.

La ventaja que tiene el Perlin Noise frente a otros ruidos clásicos (como puede ser el White Noise) es que es un ruido aleatorio, pero coherente. Es decir, de un punto a otro contiguo, apenas hay variación, lo que lo convierte en el ruido perfecto para modelar un terreno 3D.

Para tener una onda de ruido perlin final, primero se han de generar varias ondas en diferentes amplitudes y frecuencias, por ejemplo, cada onda tendrá el doble de frecuencia que la anterior, y la mitad de amplitud. Cada una de estas ondas se denomina Octava.

En un ejemplo práctico, tomando una onda senoidal *n-dimensional* de *k* octavas se realizaría:

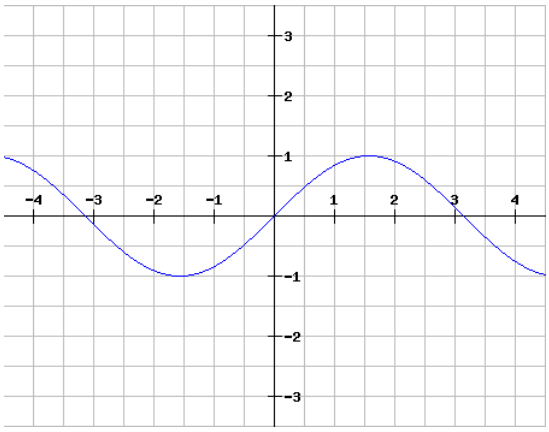


Figura 2: f(x) = Sen(x)

En la Figura 2 se observa una onda senoidal de amplitud 2 (-1 ~ 1) y longitud de onda (frecuencia) 3.

En los pasos consecuentes (figuras 2, 3 y 4) se utilizan ondas de la misma naturaleza (senoidales) pero con mayor frecuencia y menor amplitud.

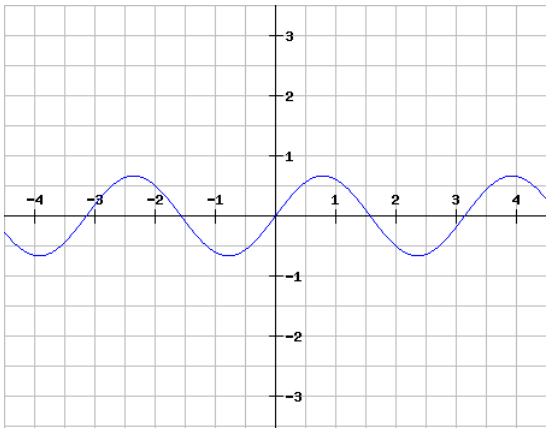


Figura 3: f(x)=Sen(x*2)/1.5

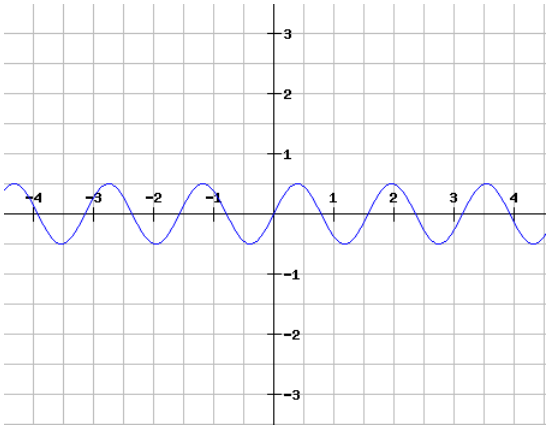


Figura 4: f(x)=Sen(x*4)/2

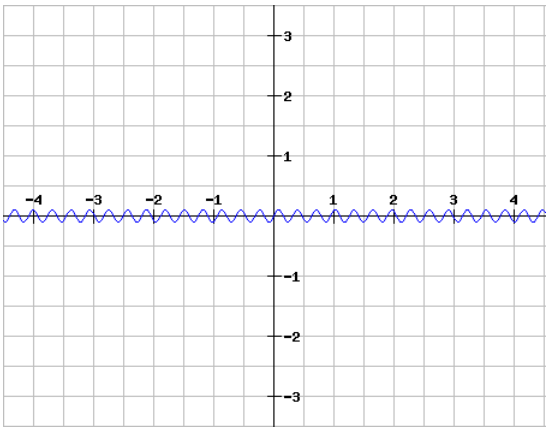


Figura 5: f(x)=Sen(x*20)/10

Usando solo cuatro octavas, obtenemos una onda perlin resultante de la suma de todas las octavas utilizadas como se aprecia en la figura 6.

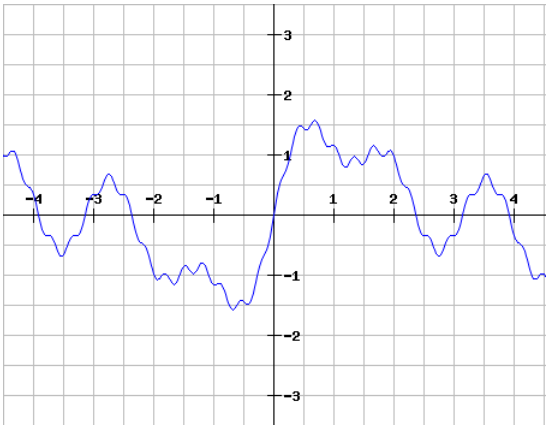


Figura 6: f(x)=sen(x) + sen(x*2)/1.5 + sen(x*4)/2 + sen(x*20)/10

En la figura 6 tenemos una onda perlin noise resultante, que se asemeja a un paisaje montañoso (en dos dimensiones).

Debido a la necesidad de crear un terreno realista, no se utiliza una onda de naturaleza senoidal, sino que se le aplica aleatoriedad eligiendo valores entre nodos (puntos en que la onda alcanza máximos o mínimos) e interpolando estos valores como se puede apreciar en la figura 6.

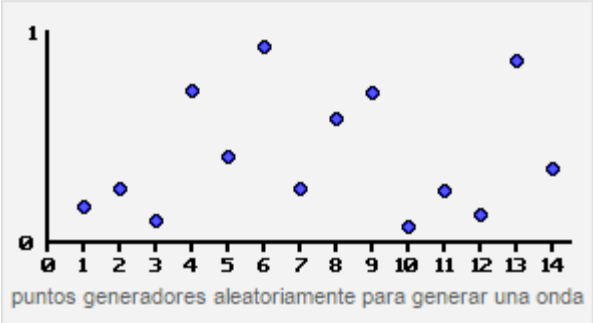


Figura 7: Ejemplo de nodos sin interpolación

Para generar el Perlin Noise que se utiliza en el plug-in se han tomado 7 octavas. La número 8 es la suma de las 7 anteriores. En estas octavas se ha aplicado una relación 2:1 multiplicando la amplitud por dos, y dividiendo la frecuencia también entre dos, ya que así se definen las octavas musicales.

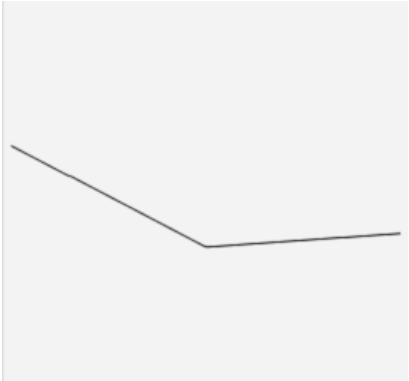


Figura 8: Primera Octava

Amplitud 2 Frecuencia 128

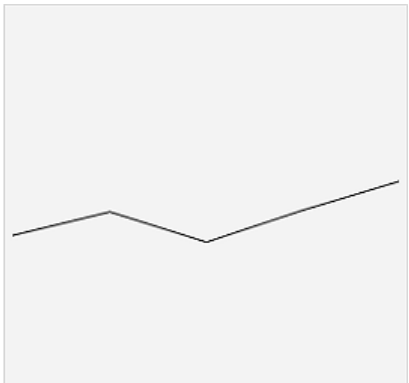


Figura 9: Segunda Octava

Amplitud 4 Frecuencia 64

Figura 10: Tercera octava

Amplitud 8 Frecuencia 32

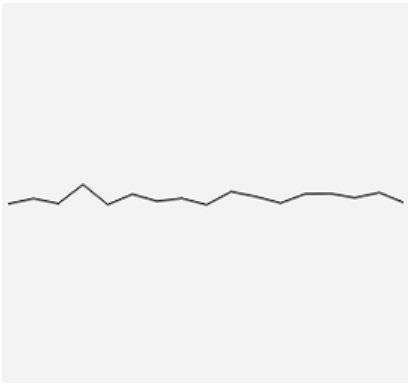


Figura 11: Cuarta octava

Amplitud 16 Frecuencia 16

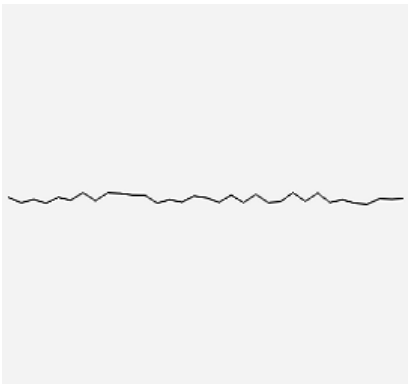


Figura 12: Quinta octava

Amplitud 32 Frecuencia 8

Figura 13: Sexta octava

Amplitud 64 Frecuencia 4

Figura 14: Séptima octava

Amplitud 128 Frecuencia 2

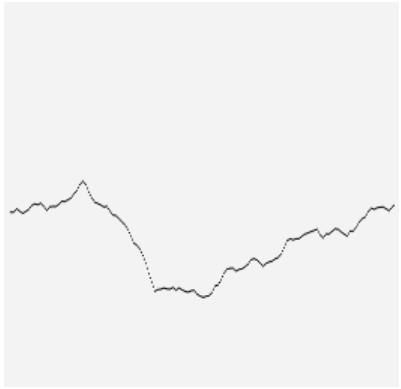


Figura 15: Ruido Perlin para el terreno.

Suma de las siete octavas.

Teniendo una onda como esta, pasamos a la generación de nuestro terreno en la clase terrain y de los atributos de terrainData.

3.3 Aplicación del Perlin Noise a la clase terrain.

Para la generación de terreno aleatoria, se ha creado una instancia de objeto de clase terrain limpia. Esto se hace mediante código:

```
terrain = new Terrain[CuadrantesEnX, CuadrantesEnZ];
```

Esa instancia realiza la misma función que crear un GameObject de clase terrain desde el editor. Con los parámetros CuadrantesEnX y CuadrantesEnZ creamos una matriz de objetos de clase terrain. Por ejemplo, una matriz de 2x2 tendría 4 terrains contiguos y coherentes.

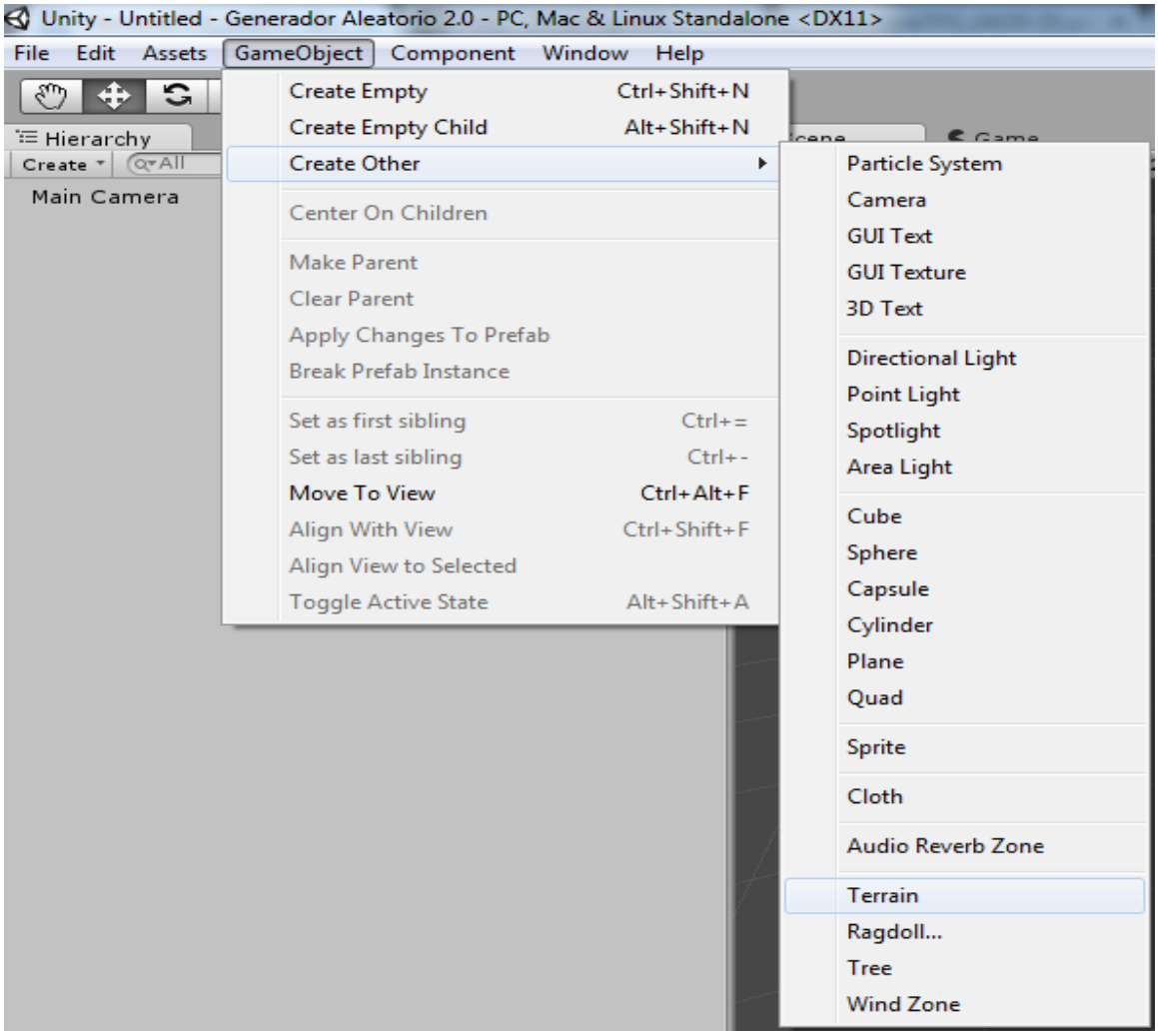


Figura 16: Creación de un terrain mediante editor

Una vez hecho esto, proceduralmente se van utilizando los métodos de la clase terrain para ir modificando los valores del terreno, editando así su estructura.

Concretamente hemos utilizado terrainData para editar todos estos parámetros.

```
TerrainData terrainData = new TerrainData();
```

```
terrainData.heightmapResolution = TamañoDelHeightmap;
terrainData.SetHeights(0, 0, alturas);
terrainData.size = new Vector3(TamañoDelTerreno, AlturaMaxima, TamañoDelTerreno);
```

Una vez editados los parámetros básicos, utilizamos una función para editar cada punto del terreno recorriendolo en forma de matriz y asignando valores que nos devuelve nuestro algoritmo de ruido.

Se comenzó utilizando la función que provee Unity de Perlin Noise propia en su API [(11)].

En un principio, se intentó desarrollar un terrain utilizando la función de la API de Unity, pero los resultados no eran terrenos realistas. En su lugar la onda producida era demasiado coherente. A su vez, no permite la suma de diferentes octavas, ya que la función de Perlin Noise itera entre 0 y 1. Al sumar varias octavas la suma resultante siempre tendía a 1, dando como resultado un terreno totalmente plano.

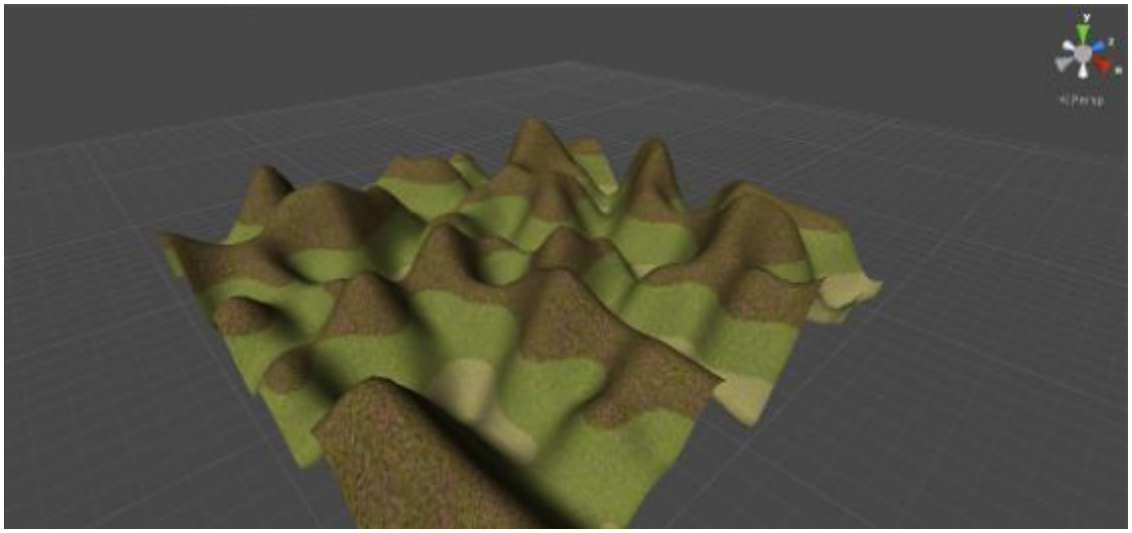


Figura 17: Generación de terreno utilizando.

Perlin Noise de Unity

Como apreciamos en la figura 17, las curvas del terreno son demasiado perfectas. Así mismo, no permite tener variaciones para que el terreno consiga zonas más llanas y zonas más montañosas.

Por este motivo, se tomó la decisión de implementar un algoritmo de Perlin Noise externo a Unity (como si de una nueva librería se tratase). Existen muchas implementaciones diferentes de este algoritmo. Para el TFG se ha utilizado una conversión del algoritmo en javascript de darkstar [(12)], miembro de la comunidad de Unity, que ha propuesto su algoritmo como una mejora del implementado en Unity.

El algoritmo ha sido traducido a C# y se le cambiaron las funciones pertinentes para adaptarlo al desarrollo del TFG.

Para conseguir una mayor naturalización, y tener zonas con llanuras y zonas con montañas, Se crean dos ondas diferentes de ruido perlin. Ambas totalmente aleatorias con una semilla diferente (a elección en la interfaz del plugin) de manera que una misma onda montañosa pueda tener diferentes tipos de llanuras y viceversa.

Una vez generadas ambas semillas, recorremos el terreno (generado como un terreno plano al inicio de la ejecución) en forma de matriz en ancho por largo. Para cada punto de nuestra matriz, asignamos un valor a una variable temporal para montaña, y otro valor para suelo. Luego combinamos ambos valores y asignamos el resultado como valor de altura en ese punto (Ver Anexo 1).

```
editTerrain(alturas, x, z);
```

El resultado es significativamente diferente al del Perlin Noise de Unity, como se puede apreciar en la figura 18:

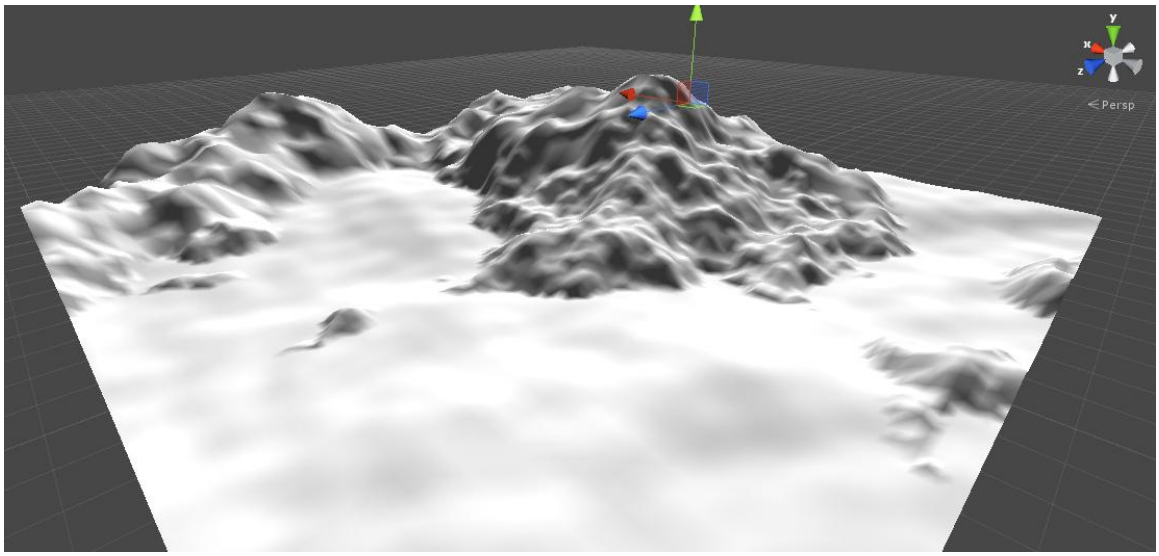


Figura 18: Terreno generado aleatoriamente.

Una vez alcanzado este punto, el terreno ya tiene forma mucho más realista y natural. El siguiente paso será darle color al terreno mediante aplicación de texturas, paso que será tratado en el siguiente capítulo.

Capítulo 4

Texturización del terreno

La texturización en Unity es otro de los procesos que más recursos consumen (tiempo), por eso automatizar esta parte es también algo vital en los objetivos del TFG.

El motor provee de un potente asistente de texturizado mediante un sistema de brochas. Este sistema nos permite elegir una textura base para todo nuestro objeto terreno, y después " pintar " con la brocha el resto de texturas sobre el mismo.

Al permitir el asistente que elijamos la opacidad y tamaño de la brocha, las texturas pueden ser definidas con todo el detalle que se desee, mientras que si son definidas por código (de manera automática) se apreciaría saltos de una textura a otra. El mayor problema de esta fase fue alcanzar una fusión entre texturas igual de "natural" que la que ofrecía el asistente, pero de manera totalmente automática. Este proceso es denominado alpha blending.

Los principales objetivos de este capítulo son la selección del método para texturizar el terreno y la aplicación de este método mediante el código del generador.

4.1 Métodos de texturización.

En Unity existen varios métodos de aplicar una textura a un objeto, algunos son particulares de cada clase de objeto, mientras que otros son métodos genéricos para cualquier tipo de objeto.

Mediante el editor que aporta Unity, el método clásico es utilizar el sistema de brocha para pintar texturas sobre el terreno.

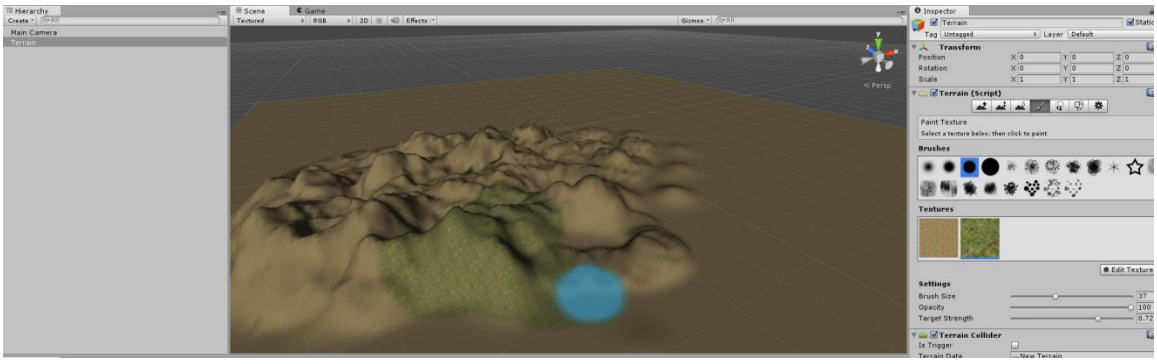


Figura 19: Pintado de texturas con el editor.

Este método de texturizar es bastante efectivo, pero a su vez, es muy lento. Lo que presenta un consumo de recursos temporales excesivo.

Como en nuestro caso estamos utilizando la clase terrain de Unity, hemos decidido utilizar uno de los métodos específicos para texturizar un objeto terrain. El splat prototype.

Este método es bastante cómodo, ya que nos permite crearnos un array de splats donde iremos almacenando las texturas que el usuario desee, para aplicar en tiempo de ejecución las texturas con alpha blending.

Una vez elegido el método, queda codificarlo.

4.2 Texturización mediante código.

Como hemos mencionado en el apartado anterior, para codificar las texturas hemos decidido utilizar el splat prototype.

Nuestro array de splats se genera de la siguiente forma:

```
array_splats = new SplatPrototype[4];
```

Otro factor interesante, es que se ha decidido aplicar la texturización en función de la altura. Según qué altura tenga el terreno en un punto, se aplicará una textura u otra.

Al mismo tiempo que se texturiza por altura, se aplica alpha blending.

Cada textura tiene un “peso” de aplicación en un punto. De manera que se puedan aplicar varias texturas a un mismo punto en el porcentaje que su peso indique.

La suma total de todos los pesos aplicados debe de ser 1.

Para poder hacer ambas cosas al mismo tiempo, se crea una función que nos calcule la altura máxima del terreno (Ver Anexo 2).

Una vez hecho esto, en la interfaz de usuario se ofrece la posibilidad de marcar lo que denominamos “Alpha cortes” que sería la altura máxima a la que se va a aplicar cada pareja de texturas. Es decir, el Alpha corte 1, sería el punto máximo donde terminaría la transición desde la textura más baja, hasta la segunda textura más baja.

Recorriendo nuestro mapa y comparando la altura con los alpha cortes, aplicamos las texturas de manera que en el punto más bajo, la textura más baja valga 1, y su peso vaya decrementando según aumenta la altura. Al mismo tiempo, el peso de la textura siguiente a esta, comienza en 0, y va incrementándose conforme incrementa la altura, hasta llegar a 1 en el punto del alpha corte (Ver Anexo 3).

Haciéndolo de esta manera conseguimos el efecto de alpha blending, ya que si pintamos las texturas directamente en función de la altura de un punto, no existe transición progresiva. En su lugar las texturas se cortan, como se aprecia en la figura 20.

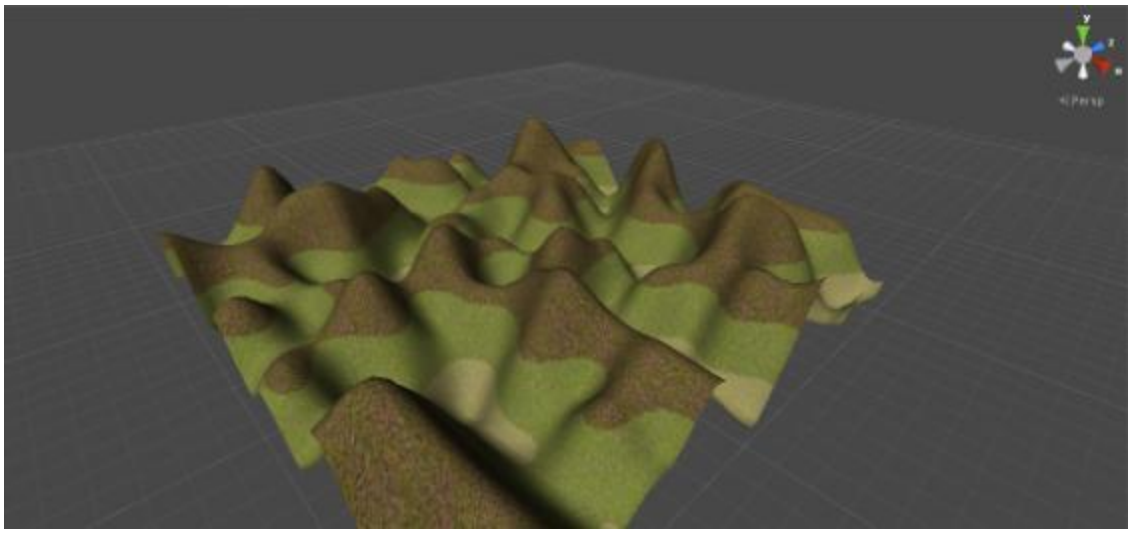


Figura 20: Texturizado por altura SIN alpha blending.

Modelado del terreno con Perlin Noise de Unity.

Por otra parte, en las figuras 21 y 22 se aprecian diferentes texturizados por altura pero aplicando alpha blending.

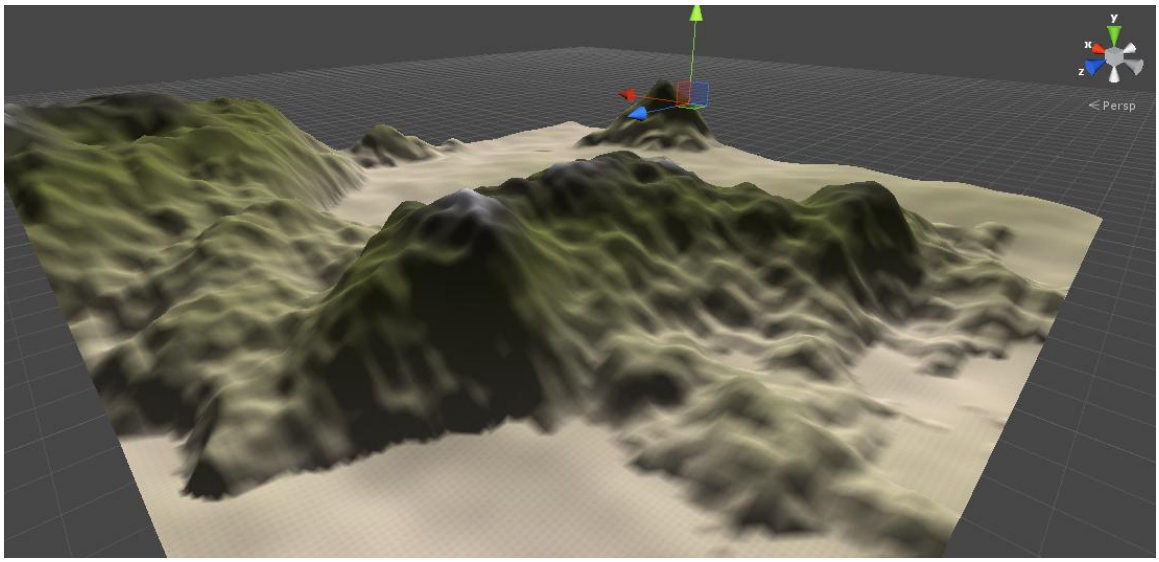


Figura 21: Texturizado por altura CON alpha blending. 4 Texturas.

Modelado del Terreno con el Perlin Noise del TFG.

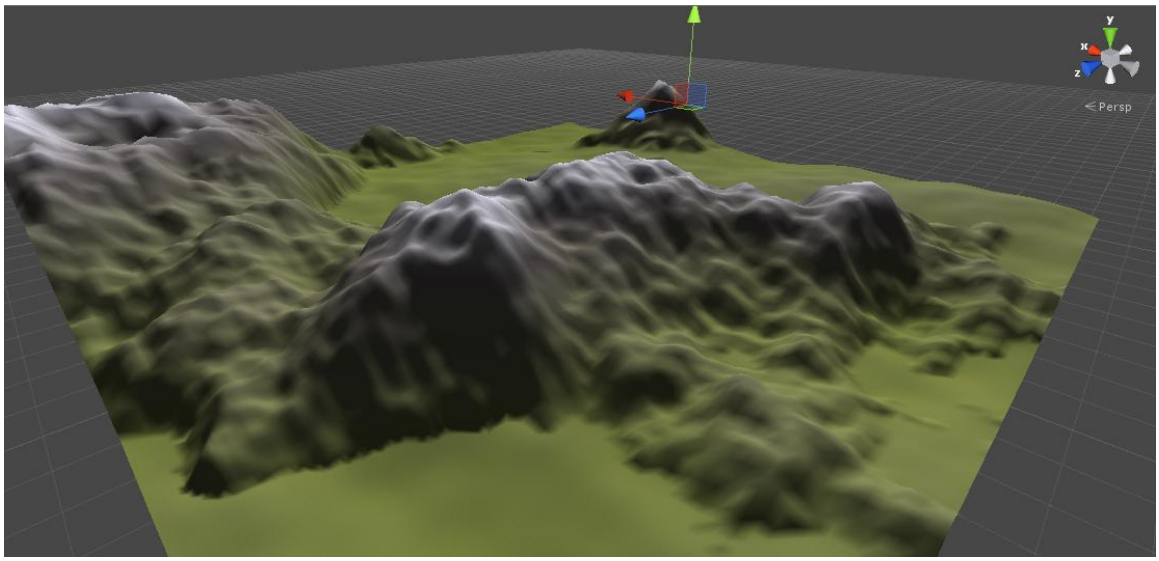


Figura 22: Texturizado por altura CON alpha blending. 4 Texturas.

Modelado del Terreno con el Perlin Noise del TFG.

El problema principal en este sistema, es que la texturización con alpha blending se hace en función de varios Alpha Cortes. Esto es solamente editable desde código (al igual que el número máximo de texturas aplicables actualmente, que es 4).

Si se desease añadir una quinta o sexta textura, habría que reutilizar el código del Anexo 3 una vez más por cada textura y crear una nueva variable pública que permita almacenar el valor del nuevo Alpha Corte.

Llegados a este punto, el terreno ya es bastante realista, pero carece de vida. La vegetación es esencial incluso en los terrenos más desérticos. Para ello se trata en el siguiente capítulo la generación de detalles varios para el mapa.

Capítulo 5

Generación de vegetación

Para evitar que el terreno parezca desértico (aunque aplicásemos texturas verdosas) habría que dotarle de cosas naturales. En este caso se ha optado por generar árboles de manera aleatoria a lo largo de nuestro terreno.

El desarrollo de árboles es un ámbito muy específico. Tanto, que hay empresas desarrolladoras encargadas solo a este aspecto. Es el caso por ejemplo de speedtree [(13)].

Para este TFG, se ha elegido tomar cualquier árbol (ya sea generado por speedtree, otras herramientas, o creado por el propio desarrollador en Blender u otros programas de modelado externos a Unity) y distribuir instancias del mismo de manera aleatoria a lo largo del mapa.

El método tradicional de Unity para dibujado de árboles, es cargar un objeto árbol (ya sea modelado en Unity o un modelador externo) y pintarlos sobre el mapa con el sistema de brocha.

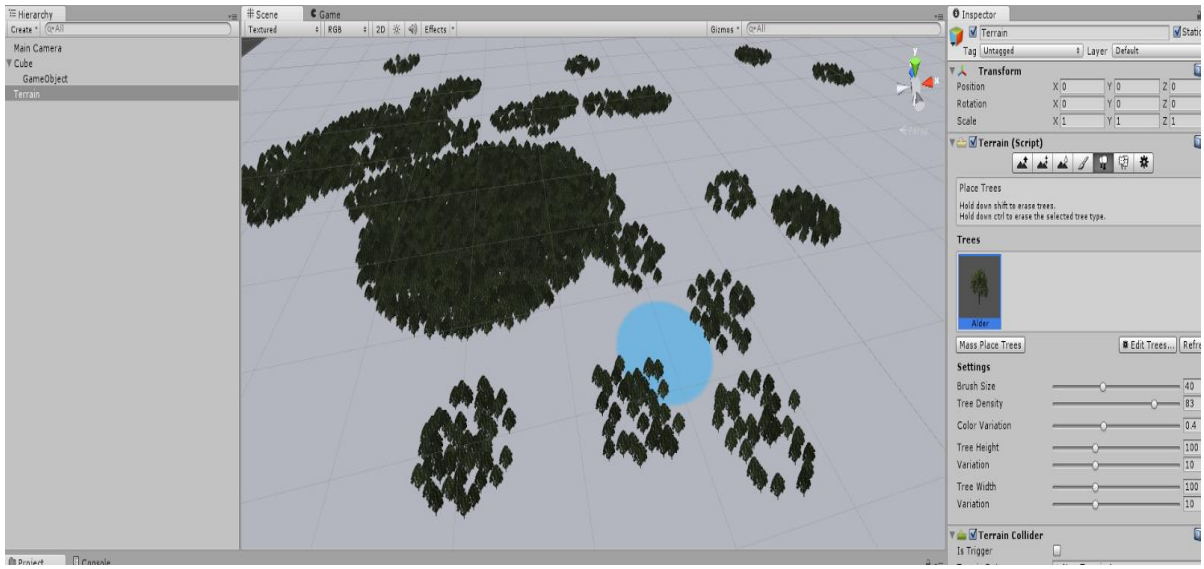


Figura 23: Pintado de árboles con el editor.

Al igual que ocurre con el método de texturizado, es un sistema algo tedioso, aunque menor que en el caso de las texturas debido a que este método no requiere de trabajar tanto con la transparencia entre texturas etc.

5.1 Generación y distribución de árboles.

Deseando reducir el tamaño del algoritmo del generador de terrenos 3D, se ha reutilizado la clase Perlin Noise previamente explicada (capítulo 3) para distribuir los árboles a lo largo del mapa.

Para instanciar árboles a elección del usuario, hemos utilizado otro atributo del terrainData, el tree prototypes.

```
arboleda = new TreePrototype[3];
```

Al igual que con las texturas, creamos un vector de tree prototypes y ahí instancia el usuario todos los que desee (el máximo actual es 3).

Una vez cargados los árboles, en tiempo de ejecución, se recorre la matriz del terreno, pero en lugar de incrementar punto por punto, incrementamos el recorrido en la matriz en N unidades, siendo N la separación mínima que debe haber entre árboles.

Una vez situados en cada punto candidato a tener un árbol, comprobamos primero el steepness en el punto (inclinación del punto, para evitar que los árboles se generen en puntos con mucha inclinación, de manera que sería irreal).

Tras validar este punto, comprobamos el Perlin Noise que hemos calculado para los árboles. Si el árbol se encuentra entre ciertos parámetros (altura máxima a la que queremos árboles, otro parámetro decidido por el usuario y menor que el valor de ruido que nos devuelve Perlin Noise) se hace una tirada aleatoria entre 0 y 3, de manera que puede seleccionarse cualquiera de los tipos de árbol instanciados, o ninguno de ellos (0) (Ver Anexo 4).

Así mismo, otros parámetros que ajustamos son:

- Distancia máxima de dibujo de árboles:
  ```
  terrain.treeDistance = 2000.0f;
  ```
- Distancia de dibujado Billboarding:
  ```
  terrain.treeBillboardDistance = 400.0f;
  ```

Estos últimos parámetros no son accesibles mediante la interfaz (por motivos de simplificación de uso) pero si están ajustados en el código, de manera que no consuma demasiado framerate en tiempo de ejecución la generación de arboledas.

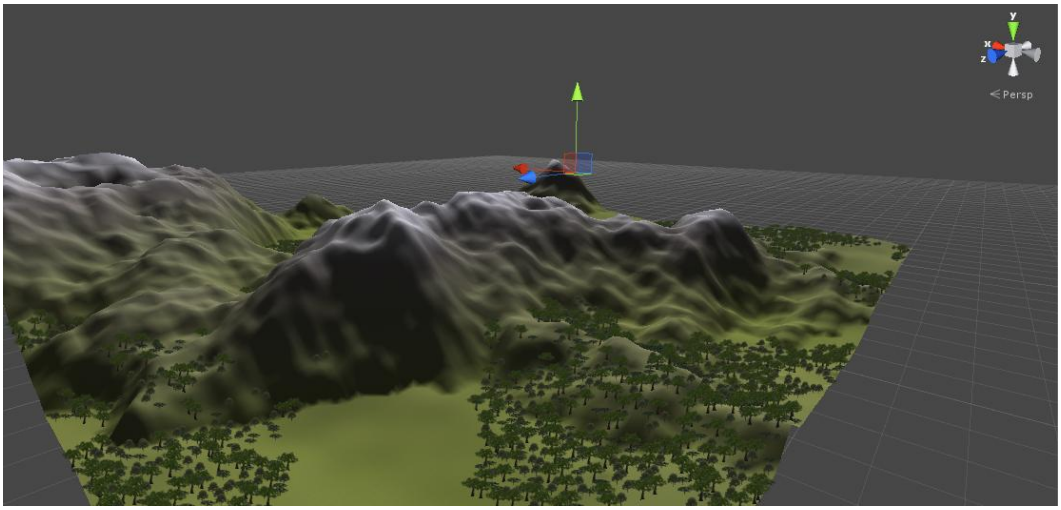


Figura 24: Arboleda al 30% de altura.

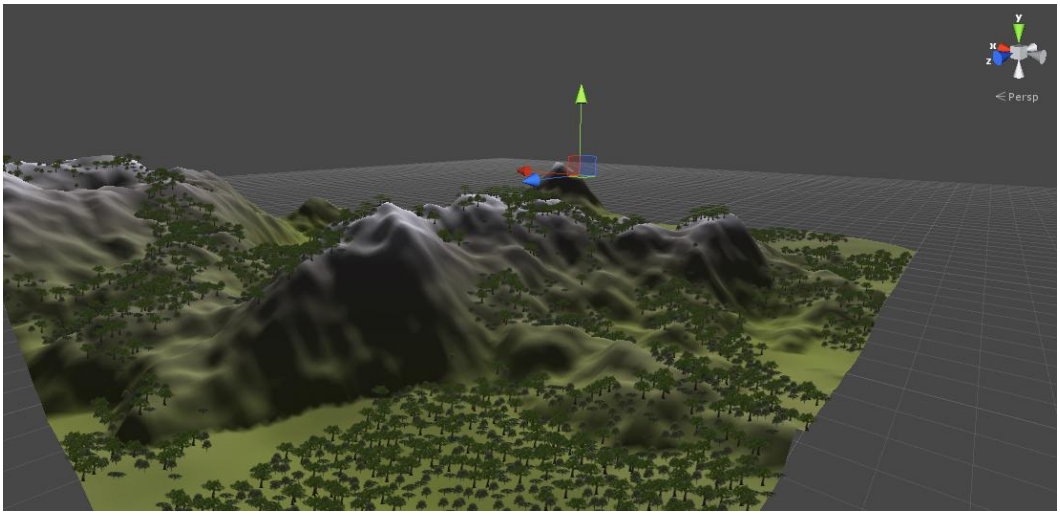


Figura 25: Arboleda al 90% de altura. Diferente semilla.

5.2 Detallado del mapa. Hierba.

Para conseguir una mayor naturalización del terreno, como se comentó en el apartado anterior, además de árboles se decidió añadir hierbas, arbustos y pequeñas flores a la generación aleatoria (siempre a elección del usuario, dependiendo de la temática del terreno).

Para esta fase hemos decidido reutilizar nuevamente nuestro Perlin Noise, de manera que distribuya texturas planas de esos “detalles”.

Estas texturas se instancian mediante el método detail prototypes de la clase terrainData. Este tipo de instancia nos permite gestionar efectos como billboarding o túnel de viento automáticamente con el motor de Unity, sin tener que realizar programación nueva para esto.

Las texturas deben de ser texturas 2D. Como se mencionó anteriormente, al ser del tipo detail dentro de terrainData, simplemente añadiendo textura 2D Unity aplica automáticamente el Billboarding.

El principal inconveniente de incluir detalles de esta manera, es que cada tipo de detalle (cada hierba, flor, etc) diferente requiere una instancia de la misma y un dibujo de mapa.

```
int[,] detalle1 = new int[TamañoDelTerreno,TamañoDelTerreno];
int[,] detalle2 = new int[TamañoDelTerreno,TamañoDelTerreno];
int[,] detalle3 = new int[TamañoDelTerreno,TamañoDelTerreno];
```

Esto significa que si incluimos 5 detalles diferentes en lugar de 3, el rendimiento del generador se ve afectado directamente, debido a que por cada detalle

ecorremos toda la matriz del terreno, generamos un Perlin Noise para ese detalle, y hacemos los cálculos por punto de si colocar o no un detalle ahí.

Así mismo también las comparaciones pertinentes para comprobar o que el ángulo del terreno no sea demasiado inclinado (de forma que no sería coherente como en el caso de los árboles) y en caso de que el punto sea candidato a tener un detalle generamos el ruido para detalles.

Si el valor del ruido supera el umbral mínimo (disponible desde la interfaz, para restringir la cantidad de detalles) se hace una tirada aleatoria de 0 a 1 (float).

A su vez, se comprueba el valor de la tirada, si es menor a ⅓ se elegirá el primer detalle, si está entre ⅓ y ⅔ se elegirá el segundo, y en caso contrario a esos dos, se elegirá el tercero(Ver Anexo 5).

Los parámetros referentes a la distancia de dibujado mínima o densidad de detalles no se han incluido en la interfaz (aunque son editables en el código) debido a que se han ajustado para no perder framerate en tiempo de ejecución.

Otros parámetros si que si se incluyen en la interaz son los referentes al túnel de viento (motor de viento de Unity):

- La velocidad de la animación (que supondría rachas de viento más constantes o menos)
  terrain.terrainData.wavingGrassSpeed = 0.4f;
- La inclinación de la animación (que equivaldría a la fuerza del viento)
  terrain.terrainData.wavingGrassStrength=0.4f;
- La cantidad de viento que deseamos:
  terrain.terrainData.wavingGrassAmount=0.2f;

Figura 26: Detalles en el terreno.
No son visibles al alejar un poco más la cámara.

Capítulo 6

Interfaz de usuario

La interfaz de usuario es una parte realmente importante del TFG. Una interfaz sencilla y accesible es necesaria para un proyecto (plugin) que busque la automatización de tareas.

En un principio se pensó en desarrollarlo como plugin integrado al menú superior de Unity, de manera que apareciera un menú contextual llamado “ Terrain Generator “ y ahí se pudiese acceder a un asistente de ventanas.

El problema con esto es que en el transcurso del TFG, Unity fue actualizado de la versión 4.X a las versiones 5 y 5.1, las cuales cambiaban el sistema de gestión de estos plugins produciendo incompatibilidades con los mismos. Por esta razón se ha optado por incluir el generador de terrenos como un script que se linka a un GameObject vacío, y se gestionan todos los parámetros mediante la ventana de script.

Una vez se haya establecido una versión de Unity 5 en la comunidad, se desarrollará el plugin correspondiente para el menú superior.

Para evitar problemas de compatibilidades, se ha implementado la interfaz mediante la interfaz de script de Unity. De esta manera, los parámetros que queremos ofrecer al usuario se declaran como públicos en el código.

Así mismo, para activar el generador, simplemente se crea un proyecto nuevo, y se aplica el script sobre un gameobject vacío, como se puede apreciar en la Figura 27.

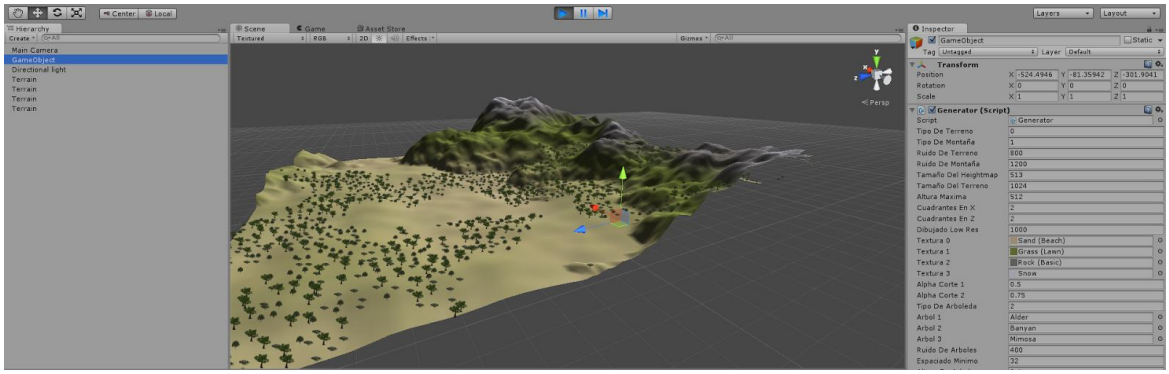


Figura 27: Interfaz de Usuario.

De esta manera, tenemos disponible en la parte derecha del editor, los parámetros comentados en los capítulos anteriores. Arrastrando texturas/árboles/detalles sobre la zona correspondiente (textura 1, por ejemplo) se carga automáticamente. El resto de parámetros son numéricos.

Capítulo 7

Conclusiones

El desarrollo del generador de terrenos aleatorios ha sido un TFG más orientado a la investigación que al desarrollo de nuevas tecnologías. Si bien Unity ofrece de por sí un potente editor de terrenos, no existen alternativas gratuitas para hacer esta tarea menos tediosa. Algunas alternativas de pago han intentado cubrir este hueco, pero ninguna tiene éxito comercial. La complejidad y poca información por parte de Unity a la hora de tratar el terreno directamente de código dificulta mucho el desarrollo de estas tareas.

Otras tecnologías como puede ser Unreal Engine 4 [(14)], tampoco facilitan la tarea de generar terrenos automáticamente en tiempo procedural, debido al enorme consumo de recursos.

Los principales problemas han sido, como se mencionó anteriormente, la falta de información. Algunos miembros de la comunidad de Unity llevan desde 2012 esperando por ciertas funcionalidades que, supuestamente, iban a ser incorporadas en las versiones 3.X. Actualmente Unity está en la 5.1 y no se han incluido aún esas funcionalidades, lo que dificulta el desarrollo de este tipo de plug-ins.

También la documentación de Unity, aunque muy buena y amplia en varios aspectos, en lo que a la clase terrain y terrainData se refiere simplemente describe lo que es cada atributo, lo cual no facilita el desarrollo.

Como conclusión final, la experiencia adquirida en este TFG ha sido muy satisfactoria. Me ha hecho desarrollar varias capacidades de investigación, así como un mejor entendimiento del desarrollo de videojuegos en general.

Capítulo 8

Conclusions

The development of the random terrain generator have been more oriented to investigation than new technologies development. Unity offers a good terrain editor, but there is no real free alternative to make this task less tedious.

Some pay alternatives have tried to solve this problem, but none have succeeded. The complexity and lack of information by Unity developers for the terrain treatment from code, make this tasks even harder.

Other technologies like Unreal Engine 4 don't easier this task either, because the big resources required.

The main problems have been, as we said before, the lack of information. Some Unity community members are still waiting since 2012 for some functionalities which should have been added to 3.X versions. Unity is now running 5.1 version and these functionalities are still missing, which makes developing this kind of plugins harder.

Also, Unity documentation is quite good, but too simple when describing terrain class and terrainData attributes.

As final conclusion, the experience earned in this project have been very satisfactory. It made me develop a lot of investigation skills, as well a better understanding of the game development in general.

Capítulo 9

Líneas de futuro

Este plugin actualmente se encuentra en una fase alpha de desarrollo. Ofrece las herramientas básicas para terrenos sencillos.

La idea final para este plugin es que sea accesible a todos desde la Asset store de Unity. Pero para publicarlo se quiere mejorar el plugin en los siguientes aspectos:

- Agua: La generación de agua mediante código es bastante confusa. Aunque se ha trabajado en estas versiones, no se llegó a ningún resultado satisfactorio por problemas de colisiones con el terreno y físicas que no funcionaban correctamente. Los planes de futuro en este aspecto es permitir la generación de agua aleatoria, de manera que pueda resultar en un terreno con muchas islas y mar por medio. Terrenos montañosos con lagunas. Terrenos con arboledas densas y ríos en medio, o combinaciones de todos ellos.
- Humanización: La mayoría de los videojuegos que utilizarían el plugin tendrían componentes humanos. Por tanto la generación aleatoria de estos ayudará en mucho el desarrollo de sus terrenos. Se desea añadir un generador aleatorio de carreteras (caminos, senderos etc, la textura a elección del usuario) así como compatibilizarlo con la generación de agua (que sea coherente, se generen puentes automáticamente etc) Del mismo modo se contempla la posibilidad de añadir un generador de edificaciones aleatorias. Al igual que con los detalles o los árboles, el usuario carga los modelos de la edificación que desee generar, y cumpliendo ciertas condiciones (inclinación del terreno, que no sea sobre agua, distancia mínima, etc) tendrá el mapa “humanizado”.
- Optimización y rendimiento: Actualmente, y para el nivel de terreno generado, el rendimiento es bastante aceptable. El problema surge cuando aumentamos el número de detalles, tipos de árboles, o si se incorporan las otras funcionalidades futuras. El Perlin Noise, aunque efectivo, es un algoritmo que consume muchos recursos computacionales, por tanto su uso para las futuras mejoras sería imposible. Se desea la incorporación de una nueva librería de ruido, o el desarrollo de una propia basada en Simplex Noise, el cual es un algoritmo mucho más eficiente.
- Interfaz de Usuario: Una vez estabilizadas las versiones 5.X de Unity, se desea convertir el script generador en un plugin real de Unity, incorporándose como un script del editor de Unity, y no del proyecto en sí. De esta manera el generador de terrenos se instalaría en Unity y estaría disponible desde el menú superior con una interfaz basada en ventanas y asistente, lo cual facilita su uso en usuarios inexpertos y lo hace más accesible en general.

Anexos

Anexo 1: Código editTerrain

```
void editTerrain(float[,] alturas, int tileX, int tileZ)
  {
     float ratio = (float)TamañoDelTerreno / (float)TamañoDelHeightmap;
     for (int x = 0; x < TamañoDelHeightmap; x++)
     {
        for (int z = 0; z < TamañoDelHeightmap; z++)
        {
           float PosX = (x + tileX * (TamañoDelHeightmap - 1)) * ratio;
           float PosZ = (z + tileZ * (TamañoDelHeightmap - 1)) * ratio;

           float montaña = Mathf.Max(0.0f, ruidomontaña.FractalNoise2D(PosX, PosZ, 6, RuidoDeMontaña, 0.8f));

           float llanura = ruidosuelo.FractalNoise2D(PosX, PosZ, 4, RuidoDeTerreno, 0.1f) + 0.1f;
           alturas[z, x] = montaña + llanura;
        }
     }
  }
```

Anexo 2: Altura máxima

```
float maxHeight(TerrainData terrainData, float max_altura)
  {
     for (int x = 0; x < tamAlpha; x++)
     {
        for (int z = 0; z < tamAlpha; z++)
        {
           float nX = x * 1.0f / (tamAlpha - 1);
           float nZ = z * 1.0f / (tamAlpha - 1);

           float height = terrainData.GetHeight(Mathf.RoundToInt(nX * terrainData.heightmapHeight), Mathf.RoundToInt(nZ * terrainData.heightmapWidth));
           if (height > max_altura)
              max_altura = height;
        }
     }
     return (max_altura);
  }
```

Anexo 3: Alpha Cortes

```
float[] splatWeights = new float[terrainData.alphamapLayers];
```

```
            float peso = 0;
            if (height <= max_altura * AlphaCorte1)
            {
                peso = height / (max_altura * AlphaCorte1);
                splatWeights[0] = 1 - peso;
                splatWeights[1] = peso;
            }
            if ((height > max_altura * AlphaCorte1) && (height <= max_altura * AlphaCorte2))
            {
                float altura2 = height - (max_altura * AlphaCorte1);
                float maximo2 = (max_altura * AlphaCorte2) - (max_altura * AlphaCorte1);
                peso = altura2 / maximo2;

                splatWeights[1] = 1 - peso;
                splatWeights[2] = peso;
            }
            if ((height > max_altura * AlphaCorte2) && (height <= max_altura))
            {
                float altura2 = height - (max_altura * AlphaCorte2);
                float maximo2 = max_altura - (max_altura * AlphaCorte2);
                peso = altura2 / maximo2;

                splatWeights[2] = 1 - peso;
                splatWeights[3] = peso;
            }
            float suma = splatWeights.Sum();
```

Anexo 4: Dibujado de árboles

```
for (int x = 0; x < TamañoDelTerreno; x += EspaciadoMinimo)
    {
        for (int z = 0; z < TamañoDelTerreno; z += EspaciadoMinimo)
        {
            float inclinación. = terrain.terrainData.GetSteepness(x,z);
            float angulo = inclinación / 90.0f;
            if (angulo < 0.5f) //Que no estén a más de 50º
            {
                float PosX = x + tileX * (TamañoDelTerreno - 1);
                float PosZ = z + tileZ * (TamañoDelTerreno - 1);

                float ruido = ruidoarbol.FractalNoise2D(PosX, PosZ, 3, RuidoDeArboles, 1.0f);
                float altura = terrain.terrainData.GetInterpolatedHeight(x,z);

                if (ruido > 0.0f && altura < AlturaMaxima * AlturaDeArboles)
                {
```

```
                TreeInstance arbolitos = new TreeInstance();
                arbolitos.position = new Vector3(x, altura, z);
                arbolitos.prototypeIndex = Random.Range(0, 3);
                arbolitos.widthScale = 1;
                arbolitos.heightScale = 1;
                arbolitos.color = Color.white;
                arbolitos.lightmapColor = Color.white;
                terrain.AddTreeInstance(arbolitos);
            }
        }
    }
}
```

Anexo 5: Dibujado de hierba

```
for (int x = 0; x < TamañoDelTerreno; x += EspaciadoMinimo)
    {
        for (int z = 0; z < TamañoDelTerreno; z += EspaciadoMinimo)
        {
            float inclinación. = terrain.terrainData.GetSteepness(x,z);
            float angulo = inclinación / 90.0f;
            if (angulo < 0.5f) //Que no estén a más de 50º
            {
                float PosX = x + tileX * (TamañoDelTerreno - 1);
                float PosZ = z + tileZ * (TamañoDelTerreno - 1);
                float ruido = ruidodetalle.FractalNoise2D(PosX, PosZ, 3, RuidoDeDetalles, 1.0f);

                if (ruido > 0.0f)
                {
                  float rng = Random.value;
                    if(rng < 0.33f)
                            detalle1[z,x] = 1;
                    else if(rng < 0.66f)
                            detalle2[z,x] = 1;
                    else
                            detalle3[z,x] = 1;
                }
            }
        }
    }
```

Bibliografía

1. Unity. Manual Unity. [Online]. Available from: http://docs.unity3d.com/Manual/index.html.
2. Unity. Forum Unity3d. [Online]. Available from: http://forum.unity3d.com/.
3. Unity. Clase Terrain, Unity3d. [Online]. Available from: http://docs.unity3d.com/ScriptReference/Terrain.html.
4. Visual Studio. Visual Studio. [Online]. Available from: https://www.visualstudio.com/.
5. Microsoft. Microsoft C#. [Online]. Available from: https://msdn.microsoft.com/es-es/library/kx37x362.aspx.
6. JavaScript. JavaScript. [Online]. Available from: https://www.javascript.com/.
7. Unity. GameObject Unity3d. [Online]. Available from: http://docs.unity3d.com/ScriptReference/GameObject.html.
8. Nvidia. [Online]. Available from: https://developer.nvidia.com/blog/5007.
9. Unity. TerrainData, Unity3d. [Online]. Available from: http://docs.unity3d.com/ScriptReference/TerrainData.html.
10. Perlin K. mrl. [Online]. Available from: http://mrl.nyu.edu/~perlin/.
11. Unity. Unity Perlin Noise. [Online]. Available from: http://docs.unity3d.com/ScriptReference/Mathf.PerlinNoise.html.
12. darkstar. Profile Unity3d. [Online]. Available from: http://forum.unity3d.com/members/darkstar.489455.
13. Speedtree. Speedtre. [Online]. Available from: http://www.speedtree.com/.
14. Unreal Engine. Unreal Engine 4. [Online]. Available from: https://www.unrealengine.com/what-is-unreal-engine-4.