Generación procedural en los videojuegos

*Memoria del trabajo de fin de grado*

Alejandro Oceja Trueba

*Dirigido por*

Domingo Gómez Pérez

# Capítulo 1: Introducción

## 1.1 Motivación

Uno de los mayores costes, tanto económicos como temporales, dentro de la industria de la creatividad y el entretenimiento digital es la creación de modelados tridimensionales. Además, a causa tanto de la incesante mejora de las capacidades de renderizado de los sistemas de entretenimiento como del aumento de la demanda por parte de los consumidores de productos de mayor complejidad y envergadura, las empresas del sector requieren de una gran cantidad de empleados y horas de trabajo, viéndose obligadas a reducir el presupuesto en otras áreas, como el desarrollo de mecánicas jugables mas avanzadas o una historia más compleja y elaborada. La generación procedural de contenido (PCG) se refiere a la creación algorítmica de contenido en un videojuego con una cantidad limitada, o indirecta de interacción por parte del usuario.

Esto permite la generación automática o semiautomática de contenido, permitiendo así la liberación de parte de los recursos destinados a estas tareas para utilizarlos en otras áreas de igual importancia.

Esto afecta aun mas a las empresas pequeñas del sector, las denominadas empresas indie, que se componen de un numero reducido de personas y cuentan con presupuestos bajos, con lo cual suelen optar por diseñar productos con gráficos pobres o menos cuidados, centrándose mas en otras mecánicas. La PCG es de especial interés para este tipo de empresas, pues les permitirá poder generar modelados de calidad sin la inversión de recursos que solo es posible para las grandes empresas del sector.

## 1.2 Trasfondo

La utilización de la generación procedural de contenido en los videojuegos existe desde los años 70, cuando se lanzó al mercado *Beneath the Apple Manor*, el primer videojuego en implementar tecnología de generación procedural para sus escenarios.

En esta época, la poca cantidad de memoria en los computadores obligaba a los creadores de videojuegos a limitar sus creaciones a pequeños escenarios, por lo que la PCG era una opción interesante para poder ampliar el contenido de los videojuegos sin necesidad de utilizar mas memoria. Posteriormente en esta época se lanza al mercado *Rogue*, un videojuego similar a *Beneath the Apple Manor*, que también utilizaba la generación procedural de contenido, y que consiguió mas fama que el primero, dando además nombre a un genero de videojuegos, el *roguelike*. En estos videojuegos, el jugador debe recorrer mazmorras que se generan de manera procedural al comenzar la partida, de forma que, en cada partida, el escenario recorrido es diferente, logrando así una cantidad de contenido imposible de alcanzar de manera manual.

Posteriormente se crearon otros videojuegos que implementaron la PCG en diferentes aspectos, por ejemplo, el videojuego de 1996 *Diablo* utiliza la PCG para generar los nombres y las estadísticas de los objetos y los enemigos que el personaje puede encontrar dentro del juego. Esto es también implementado en el videojuego *Borderlands*, en el cual existen 17,750,000 armas diferentes, todas ellas generadas utilizando algoritmos de PCG.

Otro aspecto de los videojuegos en el cual se emplea la PCG es la generación del mundo del juego. El género *sandbox* permite al jugador interactuar con el mundo del juego con total libertad, planteando al jugador objetivos básicos, como sobrevivir, o encontrar recursos. Un videojuego de esta categoría es el popular videojuego *Minecraft*, donde se utilizan técnicas de PCG para generar un mundo infinito, así como el contenido de este.

Similares a *Minecraft* podemos encontrar otros juegos que también utilizan algoritmos para generar mapas de gran tamaño, como *Terraria*, o *Civilization*, pero quizás el mayor exponente del uso de esta tecnología en los videojuegos es *No Mans Sky*, un videojuego ambientado en un universo de ciencia ficción en el que el jugador puede explorar numerosas galaxias, visitando sus planetas. Cada galaxia cuenta con millones de sistemas, y cada sistema cuenta con un numero de planetas orbitando una estrella. Además, cada planeta cuenta con su propia fauna y flora, toda ella generada de forma procedural, desde los modelados tridimensionales hasta las animaciones de cada criatura, incluso la música de el videojuego es generada utilizando la PCG.

Todo este contenido hubiese sido imposible de crear a mano, independientemente del tamaño del equipo de desarrollo encargado de la tarea.

## 1.3 Objetivo

El objetivo principal de este trabajo es aportar técnicas que permitan automatizar el desarrollo de videojuegos, más concretamente, el diseño y generación modelados tridimensionales de calidad y realismo aceptables, centrando el trabajo en la creación de una herramienta para la generación procedural de mapas realistas, además de la investigación e implementación de algunos algoritmos utilizados en la PCG.

Partiendo de ahí, nos planteamos los siguientes objetivos específicos:

Estudio de los algoritmos más comúnmente utilizados en la generación de contenido procedural, así como la implementación de algunos de estos utilizando el motor gráfico Unity.

Crear una plataforma que permita la generación de mapas tridimensionales realistas, con un nivel de detalle variable, y de forma procedural, similar al programa *World generator.*

Conseguir que los mapas generados sean interesantes desde el punto de vista del jugador, además de extenderse indefinidamente.

# Capítulo 2: Teoría de la PCG

En la generación de contenido procedural, existen numerosos métodos para abordar los problemas que estos sistemas plantean. Togelius et al. presentó en su estudio sobre la generación procedural de contenido distintas aproximaciones, que permitiesen emplazar cualquier aplicación de PCG en un punto entre dos extremos.

## 2.1.1 Online vs offline

Aquí se distingue la generación de contenido en función de si se realiza durante la ejecución del juego, *online*, por ejemplo, en el videojuego *Left 4 Dead*, la generación de enemigos en el mapa se realiza de forma procedural, variando en función de las acciones de los jugadores, o de si se realiza previamente a la ejecución del juego, o incluso durante el desarrollo de este. Como ejemplo aquí podríamos tomar una herramienta que generase mapas o modelos, sobre los que después trabajase un diseñador, para agregarlos a un juego, o una herramienta de creación de mapas dentro de un juego de estrategia en tiempo real.

Como ventaja principal de este tipo de generación *offline* tenemos que permite la creación de contenidos de alta complejidad, además de permitir un mayor control sobre el contenido final que el usuario experimentará. Como contraposición, la generación *online* permite al jugador disfrutar de una experiencia mucho mas dinámica, incluso adaptada al usuario, aprendiendo de sus decisiones y presentándole una experiencia más disfrutable.

## 2.1.2 Necesaria vs opcional

Como segunda distinción para la generación de contenido procedural, tenemos el tipo de contenido *necesario* y el *opcional.*

El contenido *necesario* es todo aquel que el jugador requiere para poder progresar a través del juego, como los mapas, mazmorras, reglas de la jugabilidad, etc., mientras que el contenido *opcional* es aquel contenido que no es necesario para la correcta progresión en juego, y por lo tanto el jugador puede escoger evitarlo, por ejemplo, armas u objetos, o lugares secundarios, a los que el jugador puede decidir no entrar, sin que ello afecte a su experiencia final.

La principal diferencia entre ambos contenidos es que el contenido *necesario* debe ser siempre correcto, o al menos tener un alto nivel de éxito, ya que generar una mazmorra que el jugador no pudiese recorrer debido a una geometría extraña, o un enemigo demasiado difícil de vencer, harían imposible para el jugador progresar en el juego.

En cuanto al contenido *opcional*, no importa si algunas veces se generase contenido de este tipo, como objetos sin utilidad o una mazmorra secundaria sin sentido, ya que el jugador siempre podrá escoger no utilizar dicho objeto o no entrar en la mazmorra.

Como ejemplo de contenido *opcional* tenemos el videojuego *Borderlands*, en el cual se generan armas de forma procedural, pero cuyo algoritmo genera en algunos casos armas que son inútiles, o simplemente inferiores a las que el jugador posee, y sin embargo esto no afecta en ningún momento a la experiencia del jugador de forma negativa, mientras que en el videojuego *Gran Turismo 5*, donde se generan de manera procedural las pistas de rally, la generación de una pista imposible de transitar para el jugador significaría un grave problema de diseño.

## 2.1.3 Aleatorio vs parametrizado

Otra distinción importante de la PCG es cuanto control se puede ejercer sobre el contenido que se va a generar. Un algoritmo puede simplemente recibir como parámetro una semilla numérica aleatoria a partir de la cual generar una mazmorra, o recibir un vector de parámetros con el numero de habitaciones, longitud de los pasillos, numero de enemigos, etc.

## 2.1.4 Estocástico vs determinista

Aquí nos referimos al grado de aleatoriedad del proceso de generación, a la cantidad de variación entre el resultado de la ejecución de un algoritmo con unos parámetros y el de la ejecución posterior del mismo algoritmo con los mismos parámetros.

Un algoritmo determinista siempre producirá los mismos resultados dados los mismos parámetros, mientras que uno estocástico variará el resultado generado.

## 2.1.5 Constructivo vs generación y prueba

En los algoritmos constructivos, el contenido se genera en una sola ejecución del mismo, como en los juegos *roguelike*, mientras que, en la generación y prueba, se genera todo el contenido, o una parte del mismo, y después se evalúa para comprobar si el resultado es lo suficientemente satisfactorio, repitiendo la generación en el caso de no serlo.

## 2.5.6 Generación automática vs generación mixta

Normalmente, en la mayoría de los algoritmos de PCG, el algoritmo recibe unos parámetros de entrada por parte del usuario, y a partir de estos, genera el contenido procedural, sin necesidad de más intervención por parte del usuario. Sin embargo, además de esta forma de generación automática, existe un tipo de generación mixta, donde algoritmo y diseñador cooperan para producir el contenido, bien el diseñador creando un nivel inacabado, para que el algoritmo rellene posteriormente los huecos de forma consistente, o bien el algoritmo genera parte del contenido, que después el diseñador puede modificar y terminar de forma más detallada.

# 2.2 Tipos de generación de contenido procedural

Dentro del amplio campo que es la generación procedural de contenido, existen diferentes técnicas para lograr el objetivo de crear un contenido que cumpla con las características deseadas, además de ser atractivo para el jugador. Este trabajo se centrará, sobre todo, en la generación procedural de paisajes y modelos geográficos de terreno realista.

Algunas de las técnicas mas utilizadas para esto son el uso de ruido, fractales, las gramáticas o los sistemas de Lindemayer.

## 2.2.1 Ruido

Las técnicas de ruido procedural se basan en funciones que mapean un determinado punto dentro de un dominio a un valor real, determinado por un parámetro pseudoaleatorio.

Antes de hablar del ruido, es importante explicar las características que lo componen:

### Frecuencia

La frecuencia del ruido es el parámetro que determina el nivel de detalle de dicho ruido. Algunas funciones tienen una frecuencia constante, mientras que otras varían dentro de un rango.

### Amplitud

La amplitud del ruido se define como la distancia entre el mínimo valor de salida y el máximo. Este valor varía a lo largo del espacio de la función, pero normalmente se suele limitar a un rango, como desde -1 a 1.

### Octavas

Se define octava como la unidad logarítmica para medir ratios entre diferentes frecuencias, siendo una octava el doble (aunque comúnmente se usan otros valores) de una frecuencia. Por ejemplo, 40 Hz es una octava de 20Hz. Comúnmente, se le añade al valor de una función de ruido para una determinada frecuencia el valor de la misma función para una octava de esa misma frecuencia, repitiendo esta operación un numero de veces, con diferentes octavas

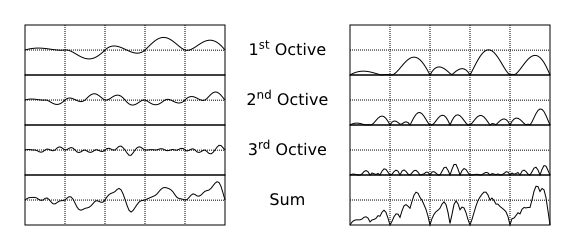


Imagen 1: Representacion de una señal de ruido Perlin al aplicarle diferentes octavas.

Este tipo de funciones tiene una serie de ventajas potenciales con respecto al resto:

-Se trata de funciones no periódicas, que se extienden en la totalidad del dominio. Esto quiere decir que pueden generar valores en un espacio de infinitas dimensiones, garantizando que no se bordes ni repeticiones en los patrones generados.

-Son funciones muy compactas, que pueden ser almacenadas con el mínimo de espacio, y que sin embargo son capaces de generar datos que de otra forma requerirían grandes cantidades de espacio de almacenamiento.

-Las funciones de generación de ruido generan valores continuos y multi-resolucion, esto quiere decir que son capaces de generar ruido a cualquier resolución deseada, sin importar lo alta que sea.

-Debido a que se trata de funciones accesibles aleatoriamente, es decir, que su tiempo de ejecución se mantiene constante sin importar el valor que se evalúa, o las anteriores evaluaciones, se puede, de forma sencilla, paralelizar la generación de datos usando este tipo de funciones.

## Algoritmos para la generación de ruido

A continuación, se exponen los tipos mas importantes de algoritmos en la generación de ruido.

### Value noise

Este tipo de algoritmos consiste en la creación de una malla de puntos, donde a cada uno de estos puntos, denominados *lattice points*, le es asignado un valor aleatorio. Hecho esto, la función de ruido devuelve, para un determinado punto, un valor basado en la interpolación de los *lattice points* a su alrededor.

Este tipo de algoritmos tiene como inconveniente que se producen cambios abruptos entre puntos cercanos, dando lugar a resultados poco realistas.

### Perlin noise

Para corregir los problemas de los anteriores algoritmos de generación de ruido, en 1895, Ken Perlin desarrollo un algoritmo similar al *value noise*, pero que, en lugar de retornar la interpolación de varios valores reales, realizaba la interpolación de varios gradientes, que eran calculados usando una función que retorna vectores bidimensionales.

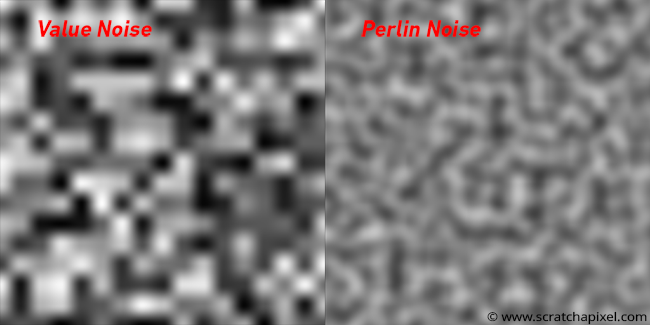


Imagen 2: Comparacion entre el ruido Perlin, primer algoritmo de Ken Perlin, y un algoritmo de Value Noise

### Simplex noise

En 2001, Ken Perlin presento una revisión de su algoritmo denominada *simplex noise*, la cual lograba las siguientes mejoras con respecto a su implementación original:

-Complejidad computacional mejorada.

-Escala a mayores dimensiones con menor coste computacional.

-Eliminación de artefactos direccionales.

-Gradientes bien definidos y rápidos de computar.

-Facilidad de implementar en hardware.