UNIVERSIDAD DE DEUSTO

NUEVO ENFOQUE PARA LA GENERACIÓN PROCEDURAL DE MUNDOS VIRTUALES VOLUMÉTRICOS

AITOR SANTAMARÍA IBIRIKA

Bilbao, 26 de junio 2014

UNIVERSIDAD DE DEUSTO

NUEVO ENFOQUE PARA LA GENERACIÓN PROCEDURAL DE MUNDOS VIRTUALES VOLUMÉTRICOS

Tesis doctoral presentada por
AITOR SANTAMARÍA IBIRIKA
dentro del Programa de Doctorado en
INGENIERÍA INFORMÁTICA Y TELECOMUNICACIÓN

Dirigida por el
Dr. PABLO GARCÍA BRINGAS
y por el
Dr. IGOR SANTOS GRUEIRO

El doctorando | El director | El director

Bilbao, 26 de junio 2014

A Belén,
por hacer realidad mi mundo virtual ideal.

Abstract

Procedural generation is a topic which includes several techniques to generate content without the intervention of a human. They are popular in many audiovisual fields, being virtual worlds generation for video games one of their main application area. Procedural virtual world enables to some video games to have an important distinguishing factor, allowing the player to explore theoretically infinite terrains with geological data.

The scientific literature handles this situation from different points of view, proposing different methods for generating virtual worlds. In spite of that, there is a lack of scientific method for generating volumetric virtual worlds. Additionally, there is not an standard method to evaluate a generator and there is not an agreement about which features should has it.

Against this background, the main objective of this research is to obtain of a valid procedural generator of volumetric virtual worlds. In order to reach this goal, this research has two sub-objectives: the enumeration of the minimum characteristics a generator should have and the proposal of a method who meets them.

First, in order to obtain the desirable features, the academical and the industrial context has been analysed collecting all the explicit and implicit aspects a procedural generator is required to have. Additionally, a set of metrics to measure these aspects has been proposed. Finally, a set of requirements has been defined determining the minimum level of each aspect a procedural generator should meet to be valid.

Next, a procedural generator for volumetric virtual worlds has been proposed. It has been designed having in mind the requirements previously established, and it is formed by two main parts: terrain generator and the playable layer generator.

On one hand, in order to generate the volumetric terrain, a new method based on fractal techniques and noise generators has been proposed. It generates volumetric sedimentary terrains based on layers. Additionally, a

method to represent volumetric terrains based on interpolation of the data has been proposed to. It requires less physical memory to store the terrain than traditional voxelized terrains.

On the other hand, the generation of the playable layer is formed by two different elements: the cave generation and the dungeon generation.

For the cave generation, we have proposed a method based on Voronoi diagrams, Delaunay triangulations and pathfinding algorithms. Its results are caves with a playability based on a distribution and interconnection of points with an special playable interest.

The dungeon generation has been tackled designing a method based on pressure techniques for room location and onn pathfinding algorithms. With our method, the designer defines the playability and the connectivity of the dungeon with graphs.

The research concludes evaluating the proposed generator with the desirable aspects, metrics and requirements previously proposed. This evaluation has been performed with an analysis of the performance and the results of the generator, remarking several signs which enables us to guess that the method meets the requirements. In spite of that, the final validation of the method requires a new set of experiments which are out of the scope of this research. We summarise those experiments in a proposal for future work.

Resumen

La generación procedural es un concepto muy amplio que comprende un conjunto de técnicas para la obtención de contenido sin necesidad de que un ser humano intervenga. Goza de una gran popularidad en las ciencias audiovisuales, aunque, en particular, es en la creación de videojuegos donde más impacto ha tenido, especialmente en la generación de mundos virtuales. Este tipo de técnicas proporcionan a algunos videojuegos un elemento diferenciador: permiten al jugador explorar mundos teóricamente infinitos, incorporando en ellos, incluso, información geológica.

El mundo científico ha abordado esta situación desde diferentes puntos de vista, proponiendo diversos métodos para generar mundos virtuales, a pesar de ello, existe una inexplicable carencia de métodos para generar terrenos volumétricos. Además, todavía no se ha llegado a un acuerdo para evaluar un generador, ni mucho menos, cuáles son las características mínimas que un generador procedural debe cumplir para ser considerado válido.

Esta situación es el punto de partida de esta tesis, cuyo objetivo principal es la obtención de un método de generación de mundos virtuales volumétricos válido, para lo que es necesario primero, el establecimiento de las características mínimas que un generador debe tener y segundo, la propuesta de un método que los cumpla.

Para obtener las características deseables, hemos analizado tanto el contexto científico como el industrial, recopilando todos aquellos aspectos, tanto explícitos como implícitos, que un generador procedural debe tener. Además, hemos desarrollado un conjunto de métricas que permiten medirlos, y establecido el nivel mínimo que cada aspecto debe cumplir para que un generador procedural sea considerado válido.

A continuación, siguiendo los requisitos establecidos en el paso anterior, hemos propuesto un generador procedural de mundos virtuales formado por dos componentes: el generador del terreno volumétrico y el generador de la capa jugable.

Para generar el terreno hemos propuesto un método basado en técnicas fractales y generadores de ruido, capaz de generar terrenos volumétricos sedimentarios. Además, hemos propuesto un método de representación de volúmenes basado en la interpolación que disminuye los requerimientos de memoria de los terrenos volumétricos tradicionales.

Para generar la capa jugable tenemos en cuenta dos puntos de vista diferentes: Por un lado, proponemos un generador de cuevas basado en diagramas de Voronoi, triangulaciones de Delaunay y algoritmos de búsqueda de caminos, cuyos resultados tienen una jugabilidad basada en la distribución de puntos de interés jugable conectados mediante galerías subterráneas.

Por otro lado, para la generación de mazmorras proponemos un método basado en técnicas de distribución de estancias por presión y en algoritmos de búsqueda de caminos. En este método, el diseñador define la mazmorra mediante grafos que determinan la conectividad entre las diferentes zonas y habitaciones. Así, se genera un entorno con una jugabilidad compleja, formado por pasillos y estancias distribuidas en diferentes pisos.

Concluimos esta investigación con la evaluación del generador mediante un análisis de sus resultados teniendo en cuenta las métricas y requisitos propuestos. Así, aunque para validar el método de una manera definitiva es necesaria la realización de una serie de experimentos que quedan fuera del alcance de esta investigación y que los proponemos como trabajo futuro, toda la información y evidencias recopiladas indican la consecución de los objetivos.

Laburpena

Prozedurazko sorkuntza kontzeptu zabala da oso. Bertan tekniken multzo batek eduki bat sortzeko jarduten du, gizakiaren esku hartze barik. Nahiz eta ospe handia izan ikus-entzunezko zientzietan, bideojokoen sorkuntzan da eragin handien duen esparrua, mundu birtualen eraketan batez ere.

Zientzia munduak, hainbat metodo proposatuz, zenbait ikuspuntutik heldu dio gai honi. Baina, hala ere, metodo gutxi dira eremu bolumetrikoen sorkuntzan. Gainera, oraindik ez da akordio batera heldu sortzaile bat ebaluatzeko, eta are gutxiago, zeintzuk diren, baliozkoa kontsideratzeko, prozedurazko sortzaile batek bete beharreko gutxieneko osagarriak.

Egoera hau da tesi honen abiapuntua; eta helburu nagusia, beraz, mundu birtual bolumetrikoak sortzeko baliozko metodo bat lortzea. Horretarako, alde batetik, beharrezkoa da sorgailuak izan beharreko gutxieneko ezaugarriak finkatzea, eta, bestetik, hauek beteko dituen metodo baten proposamena.

Behar ditugun ezaugarriak lortzeko, testuinugu zientifikoa eta industriala ikertu dugu. Horrela, prozedurazko sorgailu batek izan beharreko aspektu esplizituak eta inplizituak bilduz. Gainera, hauek neurtuko dituen metrika multzoa garatu dugu, eta prozedurazko sorgailu bat baliozkoa kontsideratzeko osagarri bakoitzak bete beharreko gutxieneko maila finkatu dugu.

Ondoren, aurreko urratsean finkatutako baldintzei jarraiki, bi osagarriz eratutako mundu birtualen prozedura sorgailua proposatu dugu. Bi osagarriok eremu bolumetrikoa eta jolaserako geruzaren sorgailua dira.

Eremua sortzeko, eremu bolumetriko sedimentarioak sortuko dituen teknika fraktaletan eta zarata sorgailuetan oinarritzen den metodoa proposatu dugu. Horretaz aparte interpolazioan oinarritutako eremu bolumetriko tradizionalek behar duten memoria murrizten duen bolumen errepresentazio metodoa ere proposatu dugu.

Jolaserako geruza sortzeko, ikuspuntu bi kontuan hartu ditugu:

Alde batetik, koben sorgailu bat proposatu dugu, Voronoiren diagrametan, Delaunayren triangulaketetan eta bideak bilatzeko algoritmoetan oinarritzen dena; eta emaitzek lurpeko galeriez konektatutako jolaserako interes puntuen distribuzioan oinarritutako jolasgarritasuna dutelarik.

Bestetik, ordea, ziegen eraketarako, bideak bilatzeko algoritmoetan eta presio gelen distribuzio tekniketan oinarritutako metodoa proposatu dugu. Metodo honetan, diseinatzaileak gelen eta eremuen arteko lotura konplexua determinatuko dituen grafo bidez definituko du ziega.

Ikerketari amaiera emango diogu sorgailuaren ebaluazioarekin, kontuan hartuz aldez aurretik proposatutako metrika eta baldintzak. Hortaz, bildutako azkeneko informazioak helburuen erdiestea adieraziko du. Hala ere, metodoa behin-betiko era batean balioesteko, etorkizuneko lan bat proposatzen dugu, ikerketa honetatik at geratzen diren esperimentuen beharra baitago.

Agradecimientos

Antes de entrar en materia me gustaría expresar mi agradecimiento a todas aquellas personas que han hecho posible esta investigación doctoral.

En primer lugar, quiero agradecer a mis directores de tesis y a la vez compañeros, los doctores Pablo García Bringas e Igor Santos Grueiro, el apoyo personal y profesional recibido durante estos años. Gracias Pablo, por haberme permitido iniciar esta aventura. Gracias Igor, por enseñarme todo lo que he necesitado para llevarla a cabo.

Gracias al increíble grupo humano que compone el *S3lab (Laboratory for Smartness, Semantics and Security)*, uno de esos sitios donde trabajar es un privilegio. Entre todos ellos me gustaría destacar a Xabier Cantero, gracias por ser tan genial personal y profesionalmente; y a Asier González y a Jaime Devesa, gracias por todos los buenos momentos que solo unos verdaderos amigos pueden proporcionar. Los ratos que pasamos los cuatro juntos han aportado a esta investigación mucho más que todas las publicaciones científicas en las que se basa. Cómo olvidar a Xabier Ugarte, por acordarse de mí para aquella oferta de trabajo cuando languidecía en una consultora; a Mikel Salazar, por sus continuos aportes; a Borja Alonso, por enseñarme que *pushear* no es siempre la mejor opción; o a Sergio Huerta, por presentarme a Voronoi. Tampoco quiero olvidarme de otros integrantes del *S3lab* que, de alguna manera, también han influido en esta investigación: Borja Sanz, Carlos Laorden, Javier Nieves, Juan López de Uralde, José Gaviria, Félix Brezo, Agustín Zubillaga, Iker Pastor, Sendoa Rojas, Patxi Galán, Iván García y Guillermo Mariscal.

Eskerrik asko, lagunak!

También quiero aprovechar estas líneas para agradecer a la Universidad de Deusto, a Deustotech y a la *Università degli studi di Bergamo* el permitirme realizar una estancia doctoral europea en una ciudad tan única como Bérgamo. En especial, me gustaría agradecer al Dr. Giuseppe Psaila su apoyo durante la estancia.

Grazie mille!

Por supuesto, quiero agradecer a mi familia todo lo que me ha aportado. Gracias *Aita*, por la sonrisa que se te escapa cuando hablas de mí y por enseñarme lo que significa dar siempre el máximo. Gracias *Ama*, por confiar siempre en mí y por hacer mi bienestar tu absoluta prioridad. Aquellas tardes jugando a videojuegos los dos juntos son uno de mis mejores recuerdos. Como veis, he aprovechado bien la *Nintendo* que me regalasteis y sobre la que tantas ganas puse para saber cómo funcionaba. Gracias Irati, y ánimo, que pronto serás tú la doctora, aunque de bata blanca y estetoscopio. Gracias *Aitite* y *Amama*, por toda la atención y cariño que me habéis dado, y por dejar que «arreglase» todas las radios que teníais en casa. Gracias Edurne, por ser mi otra madre. Gracias Matías y Mari Carmen, por vuestro apoyo e interés por este trabajo. Gracias Xabi, por la rapidez de la traducción al Euskera. Y gracias también a todos los que faltan en esta lista, por estar ahí cuando os he necesitado.

Eskerrik asko, familia!

Finalmente, quiero dar las gracias especialmente a la persona que da sentido a todo esto. Gracias Belén, por quererme como me quieres, por apoyarme incondicionalmente en todo lo que hago y por ser mi compañera de viaje en la vida sea cual sea el camino.

Como apunte final, me gustaría disculparme con todas aquellas personas que no he mencionado. Ya sabéis que la memoria no es una de mis virtudes, por lo que si creéis que deberíais aparecer en estas líneas, gracias a vosotros también.

Este trabajo tiene un poco de todos vosotros.

Eskerrik asko denoi!

Índice general

Índice de figuras

Índice de tablas

«¿Que los videojuegos son malos? Eso mismo decían del rock and roll.»

Shigeru Miyamoto (1952-presente)

«Esta ciudad no se levantó con hormigón y acero; se levantó con ideas!»

Andrew Ryan (*Bioshock*, 2K, 2007)

CAPÍTULO

1

Introducción

La popularidad e importancia de la industria de los videojuegos ha crecido de manera meteórica desde sus inicios en los años 60, pasando de ser un pequeño sector dentro de la industria del ocio a facturar varias decenas de miles de millones de dólares al año [AMO12]. Además, en la última década ha dejado de considerarse una opción de ocio alternativa para convertirse en un tipo de entretenimiento aceptado por toda la sociedad.

De manera paralela a este crecimiento económico y social, esta industria ha sufrido también una evolución técnica muy acentuada. La evolución sufrida en el *hardware* y el *software* en las últimas décadas ha revolucionado la informática gráfica, influyendo en gran medida en el sector de los videojuegos. Esto ha repercutido en aspectos tales como la capacidad de gestión y renderización de escenas enormes muy detalladas. Además, este crecimiento ha causado que grandes empresas y diseñadores de prestigio dediquen cada vez más esfuerzo y recursos al diseño y desarrollo de videojuegos.

Con un solo vistazo a su aspecto es posible hacerse una idea de lo supone esta evolución. En la figura 1.1 podemos ver tres juegos de diferentes épocas que nos permiten apreciar el cambio que ha sufrido el sector. Entre el *Pong*, en la subfigura 1.1(a), y el *Catacomb 3D* en la 1.1(b), hay 20 años de diferencia; los mismos que entre el *Catacomb 3D* y el *The Elder Scrolls V: Skyrim*, en la subfigura 1.1(c).

Como vemos en la figura 1.1, todos los aspectos han evolucionado de una forma

(a) *Pong* - 1972 [Ata72] (b) *Catacomb 3D* - 1991 [ID 91]

(c) *The Elder Scrolls V: Skyrim* - 2011 [Bet11]

Figura 1.1: Evolución de los videojuegos.

radical, aunque si nos fijamos, podemos observar que el entorno en el que se desarrollan los juegos es uno de los elementos que más ha cambiado: en la subfigura 1.1(a) podemos apreciar un campo de tenis formado simplemente por un fondo negro y una red blanca representada por una línea; en la subfigura 1.1(b) ya apreciamos un entorno más complejo, donde podemos diferenciar paredes, suelos y techos, aunque sin mucho detalle aún; y en la subfigura 1.1(c) podemos ver un entorno muy complejo formado por montañas, árboles, rocas y plantas extremadamente detalladas.

Estos entornos en los que se desarrollan los videojuegos u otras aplicaciones gráficas son comúnmente conocidos como mundos virtuales.

1.1 Mundos virtuales

Los mundos virtuales son aquellos mundos donde los videojuegos desarrollan su acción, y son una representación gráfica de un entorno que los diseñadores han definido. Estos entornos pueden ser desde una mazmorra medieval hasta un campo de batalla de la Segunda Guerra Mundial, pasando por el enésimo planeta extraterrestre en el cual el

héroe debe sembrar su particular caos.

Dependiendo del tipo de juego y de sus características, el mundo virtual puede cambiar radicalmente, aunque es habitual que tenga varios elementos comunes, como un terreno o una serie de entidades interactivas que lo pueblan y lo hacen interesante para el jugador.

1.1.1 Terreno

Tradicionalmente, los terrenos utilizados en videojuegos están compuestos por una matriz de alturas, comúnmente conocida como *heightmap*[1]. Este tipo de terrenos son muy útiles debido a su simplicidad, pero tienen muchas limitaciones, como la imposibilidad de contener cuevas y salientes rocosos, o de gestionar deformaciones complejas.

Por otro lado, existen los terrenos volumétricos, que almacenan información de su interior (normalmente datos geológicos sobre los materiales que los forman y la disposición en la que se encuentran). Estos terrenos permiten definir cualquier tipo de estructura, y son fácilmente modificables y deformables en tiempo real. Aunque es posible representar dichos terrenos mediante diferentes métodos, tradicionalmente se han representado mediante *voxels*[2], de los que podemos ver un ejemplo en la figura 1.2.

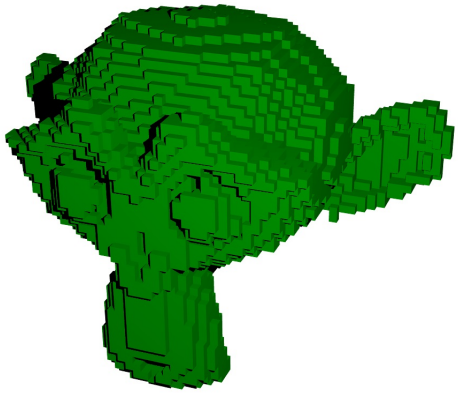


Figura 1.2: Suzanne, la mascota de *Blender* [Ble95], creada mediante *voxels*.

El primer juego en utilizar un terreno basado en *voxels* fue *Comanche Maximum*

[1] Un *heightmap* o mapa del alturas es una matriz bidimensional que contiene las alturas de la superficie de un terreno.

[2] Un *voxel* es la unidad mínima en las que puede dividirse un volumen. Se encuentran posicionados en el espacio respetando una matriz tridimensional regular.

Overkill [Nov92] de 1992, aunque solamente utiliza *voxels* para visualizar la superficie de una manera realista y no para manejar información volumétrica. A pesar de ello, el juego que ha popularizado la utilización de *voxels* y de información volumétrica del terreno en los últimos años ha sido *Minecraft* [Moj11].

1.1.2 Aspectos jugables

El interés que un mundo virtual genera en el jugador viene dado, en gran medida, por el conjunto de aspectos jugables que lo hacen interactivo. Estos aspectos son completamente dependientes del tipo de juego para el cual se construyen. Por ejemplo, en un juego de exploración es necesario tener mazmorras construidas con una estructura específica para que el jugador las descubra, así como puzles correctamente distribuidos y diseñados, y tesoros a recoger; en una aventura es necesario poblar el mundo con personajes con los que poder interactuar y obtener pistas para avanzar en la trama; y en un juego de acción es necesario determinar las zonas de aparición de los diferentes enemigos y donde poder encontrar nuevas armas.

Estos aspectos son los principales responsables de definir el tipo de videojuego que se quiere construir, así como de conseguir que un mundo virtual sea jugable y divertido, y de darle mayor profundidad que la que se ve a primera vista.

1.1.3 Otros aspectos

Existen muchos otros elementos de los mundos virtuales que no hemos descrito, como las ciudades y sus edificios, la vegetación o las redes de carreteras y caminos que interconectan las diferentes zonas. Estos elementos aportan profundidad, coherencia y realismo al mundo virtual.

1.2 Generación procedural de mundos virtuales

Como ya hemos visto, la evolución sufrida por el sector de los videojuegos ha causado, entre otras cosas, que los mundos virtuales en los cuales se desarrollan sean cada vez más detallados. Además, algunos juegos necesitan información de la composición interna de sus volúmenes requiriendo una enorme cantidad de datos para sus mundos virtuales.

La construcción del mundo virtual es una tarea repetitiva, que se realiza de manera

iterativa definiendo cada vez más cada uno de sus aspectos. Este proceso se alarga en el tiempo y se hace especialmente tedioso cuanto más grande y detallado sea el mundo. Además, debe llevarse a cabo por un equipo multidisciplinar formado por perfiles que requieren habilidades técnicas muy específicas: diseñadores que determinen el aspecto del mundo, diseñadores de niveles que busquen estructuras adecuadas para que sea divertido de jugar, modeladores para modelarlo gráficamente y programadores para dotarlo de vida y movimiento.

Ante esta situación, la solución que ha adoptado la industria de los videojuegos es la utilización de algoritmos de generación procedural de mundos virtuales.

«Generación procedural» es un término usado en la creación de elementos multimedia que engloba cualquier tipo de generación de contenido de manera automática o semiautomática. Existen multitud de métodos que utilizan y/o combinan diferentes técnicas, como construcción de fractales o algoritmos evolutivos. Este tipo de generación de contenido está muy extendido para crear elementos tales como modelos, música o animaciones, aunque hay casos en los que se ha llegado a utilizar, incluso, para obtener galaxias enteras con complejos sistemas políticos interplanetarios [BB84].

Algunos autores han enumerado aquellos aspectos críticos que determinan la validez de todo generador procedural [DP10, Sau06, OSKL05]. Estas directrices son específicas para la generación de terrenos procedurales, aunque es fácil sintetizarlas y extrapolarlas para generadores de mundos virtuales completos, siendo estas las siguientes:

- **Innovación:** Debe generar elementos que no espere el diseñador.
- **Estructura:** Sus resultados deben estar estructurados, es decir, que no sean simple ruido aleatorio. Debe generar estructuras reconocibles que se integren entre sí.
- **Interés:** Debe generar mundos interesantes para el jugador.
- **Velocidad:** Debe ser suficientemente rápido para que los resultados estén disponibles cuando se los necesita.
- **Usabilidad:** Los parámetros de entrada deben permitir un máximo control en el resultado final, así como estar directamente relacionados con el elemento a generar y no con el algoritmo utilizado. Además, deben ser los mínimos posibles.
- **Ampliabilidad:** Debe generar diferentes tipos de mundos con cambios mínimos o, directamente, con simples modificaciones en los parámetros de entrada.

- **Escalabilidad:** Debe generar terrenos con suficiente detalle a diferentes escalas.

En el capítulo 3 analizamos estas características de una manera más exhaustiva.

1.2.1 Generación supervisada

A la hora de generar mundos procedurales existen dos enfoques diferentes, generación supervisada y generación no supervisada.

La generación procedural supervisada es el proceso de generar el mundo virtual mediante métodos procedurales en los cuales interviene la figura del diseñador. Reduce drásticamente el tiempo necesario para construir el mundo y permite construir entornos muy detallados con facilidad. Además, es posible corregir cualquier aspecto generado automáticamente que no tenga las características deseadas. El principal aspecto negativo de este método es que el usuario final del videojuego no percibe ninguna diferencia respecto a una construcción tradicional del mundo virtual.

Este tipo de generadores procedurales tiene una serie de problemas históricos que llevan arrastrando desde el mismo momento de su creación. Smelik *et al.* [STdKB08] identifican estas deficiencias y proponen una serie de cualidades que todos los *frameworks* de generación de terrenos deben cumplir. Estos requisitos atienden a términos de usabilidad del generador, visualización del terreno, facilidad para determinar los accidentes geográficos principales, etc.

1.2.2 Generación no supervisada

La generación procedural no supervisada es el concepto opuesto a la supervisada. En este enfoque, la generación se realiza de manera completamente automática, ya que limita la influencia del diseñador a un conjunto de parámetros iniciales y elimina completamente su intervención durante la ejecución del proceso.

Llevando al límite la idea de generación procedural no supervisada, es posible conseguir una característica especial que puede explotarse para crear juegos muy particulares, en los que la generación se realiza en tiempo de ejecución. Esto es, en cada partida se obtiene un mundo virtual diferente, siendo posible incluso generarlo en tiempo real según el jugador lo va descubriendo. Así, los videojuegos que utilizan esta característica llevan la exploración y la rejugabilidad a límites muy superiores a los obtenidos con mapas predefinidos.

Este método de generación de terrenos procedurales tiene tres desventajas importantes frente a la generación supervisada:

- El diseñador no puede aportar nada durante el proceso de generación, por lo que todo el conocimiento debe proveerse en los parámetros de entrada.
- El terreno generado debe ser directamente usable, es decir, el generador debe ser capaz de determinar la posición y características de todas las zonas, entidades y estructuras jugables necesarias para la ejecución del juego.
- La velocidad de ejecución del algoritmo es crítica, ya que la generación se lleva a cabo en el sistema del usuario.

Debido a estas desventajas, los juegos con este tipo de generación procedural de mundos virtuales suelen tener menor variedad de entornos y un nivel de detalle mucho más bajo que los generados con métodos supervisados o los modelados de manera tradicional.

1.3 Formulación del problema

La generación procedural es un campo muy extendido en la industria, en particular, para la generación de mundos virtuales, aunque existe una inexplicable carencia en la literatura científica para la generación procedural de terrenos volumétricos.

Debido ello, los videojuegos o simulaciones que utilizan terrenos volumétricos generados proceduralmente presentan métodos de generación primitivos, poco desarrollados y no validados científicamente, por lo que dejan demasiados aspectos al azar. Estos terrenos suelen estar basados principalmente en generadores de ruido Perlin [Per85] simple. Además, estos generadores no están diseñados teniendo en cuenta los aspectos enumerados en la sección 1.2, ya que se centran exclusivamente en el problema individual de cada uno sin aportar una solución global y personalizable.

Asimismo, como veremos posteriormente (ver sección 2), la mayor parte de los métodos existentes en la literatura para generar terrenos se centran en el aspecto visual y no en la capa jugable que los mundos virtuales necesitan para ser utilizables en una aplicación interactiva. La generación de esta capa jugable es un elemento crítico en los algoritmos no supervisados, ya que no existe la figura del diseñador que después crea esta capa.

Este problema se ve agravado por la falta de unanimidad en los métodos para la validación de los generadores procedurales, ya que no existe un único método ni criterio para evaluarlos.

En resumen, para el desarrollo de esta rama del sector es crucial disponer de herramientas mediante las cuales poder extender y mejorar las características de los terrenos volumétricos generados proceduralmente que se están utilizando a día de hoy en la industria. Además, es necesario disponer de mecanismos que permitan evaluar, validar y cuantificar la diferencia de manera objetiva entre este tipo de generadores.

1.4 Hipótesis y objetivos

Frente a esta situación y con el problema identificado y definido, formulamos la siguiente hipótesis:

> «*Es posible, empleando una combinación de técnicas espaciales y de inteligencia artificial, generar mediante métodos no supervisados mundos virtuales volumétricos que cumplan todos los requisitos, tanto explícitos como implícitos, propuestos tanto por la comunidad científica como por el entorno industrial. Además, es posible sintetizar este conjunto de requisitos formalmente, obteniendo métricas objetivas para evaluarlos.*»

Realizamos la formulación de esta hipótesis con el objetivo de probar que es posible obtener un método no supervisado de generación de mundos virtuales con terrenos volumétricos que cumpla con todos los requisitos que se proponen en la literatura, así como con los autoimpuestos por el entorno industrial que sean comunes a los diferentes generadores.

Para conseguir demostrar esta hipótesis hemos establecido el siguiente objetivo principal:

Objetivo principal 1. *Desarrollar y evaluar mediante los requisitos, tanto explícitos como implícitos, presentes en la literatura e industria, un método genérico no supervisado para la generación procedural de mundos virtuales volumétricos.*

Para conseguir alcanzar dicho objetivo planteamos una serie de objetivos específicos, los cuales detallamos a continuación:

Objetivo específico 1. *Identificar y sintetizar los requisitos presentes en la literatura y en la industria para los generadores procedurales.*

Objetivo específico 2. *Desarrollar y evaluar mediante los requisitos previamente identificados, un método de generación procedural de terrenos volumétricos y sus estructuras jugables.*

El primer objetivo específico comprende un análisis de las características deseables que deben cumplir los generadores de terrenos y su capa jugable, así como su posterior síntesis en un conjunto de requisitos cuantificables. El segundo objetivo específico es el desarrollo y validación del método de generación de los mundos virtuales volumétricos cumpliendo los requisitos identificados en el primer objetivo específico.

A fin de lograr la consecución de estos objetivos específicos hemos definido los siguientes objetivos operacionales.

Objetivo operacional 1. *Identificar los aspectos críticos para los generadores procedurales de mundos virtuales presentes en la literatura.*

Objetivo operacional 2. *Identificar los aspectos críticos para los generadores procedurales de mundos virtuales presentes en el entorno industrial.*

Objetivo operacional 3. *Asignar un conjunto de métricas a cada uno de los aspectos críticos identificados previamente, así como el nivel requerido para cada aspecto, formando un conjunto de requisitos cuantificables para los generadores procedurales de mundos virtuales.*

Objetivo operacional 4. *Desarrollar un método procedural de generación de mundos virtuales volumétricos atendiendo a los requisitos previamente identificados.*

Objetivo operacional 5. *Evaluar el método desarrollado utilizando el conjunto de requisitos establecido mediante las métricas.*

Los dos primeros objetivos operacionales tienen como meta descubrir los aspectos críticos para los generadores procedurales de mundos virtuales, mientras que el tercero hace referencia a la asignación de una métrica a dichos aspectos, así como al nivel

mínimo indispensable dentro de dicha métrica para considerar válido el método de generación. Por su parte, el cuarto objetivo comprende el diseño del método en sí mismo. Finalmente, el quinto objetivo operacional tiene como meta la evaluación del método propuesto mediante las métricas y requisitos establecidos previamente.

1.5 Metodología de investigación

Con el fin de alcanzar de una manera satisfactoria los objetivos de la investigación llevada a cabo en esta tesis (expuestos en la sección 1.4), seguimos una metodología que minimice los problemas históricos en la evaluación de resultados en el área de los gráficos por ordenador, incluyendo el campo de la generación procedural.

Para ello hemos desarrollado una metodología de investigación basada en la generación de prototipos que simulen aplicaciones reales de la industria.

1. **Adquisición de conocimiento:** En primer lugar, analizamos el contexto sobre el cual encaja nuestra investigación. Para ello estudiamos en profundidad el estado del arte de la generación procedural de mundos virtuales, así como su situación actual en la industria. Esto nos permite tomar la perspectiva adecuada para afrontar la investigación, así como entender correctamente el encaje de esta tesis en su entorno.

2. **Identificación y formulación del problema:** La primera etapa de la investigación es la identificación de las características necesarias para los generadores procedurales de mundos virtuales, así como las métricas para evaluarlas, lo que nos permite más adelante realizar una primera evaluación de los métodos propuestos. Además, en este paso realizamos un acercamiento práctico al método, especificando los requisitos y permitiendo focalizar nuestros esfuerzos en los aspectos adecuados.

3. **División del problema y diseño modular de la solución:** Una vez definido el problema mediante un conjunto de requisitos, diseñamos la solución de una manera modular y proponemos soluciones para la generación de los diferentes aspectos que componen un mundo virtual.

4. **Validación mediante prototipos:** En este punto de la metodología construimos un prototipo funcional de cada módulo propuesto. De esta manera, evaluamos los método en un entorno que simula la realidad de la industria de las aplicaciones gráficas interactivas. Para cada uno de estos módulos realizamos una serie de

experimentos en los que recopilamos información relevante, como el tiempo que requiere para ejecutarse.

5. **Análisis de resultados:** Una vez realizadas las pruebas mediante los prototipos analizamos los resultados obtenidos mediante la comparación con los requisitos establecidos en pasos previos de la investigación utilizando las métricas definidas. Dependiendo de los resultados de dicho análisis, el proceso vuelve a diferentes pasos anteriores para modificar aquellos aspectos que resulten mejorables.

6. **Difusión de resultados:** El último paso de la metodología propuesta cierra el ciclo natural del proceso científico clásico dando a conocer el método y sus resultados a la comunidad científica. Este proceso se realizará mediante la publicación de la investigación realizada en esta tesis en conferencias y revistas científicas internacionales. Esto nos permite evaluar la importancia de la investigación realizada, así como la puesta en común de conocimiento con otros expertos del área.

1.6 Estructura de la tesis

El resto de capítulos de esta tesis se encuentran estructurados de la siguiente manera:

Capítulo 2: Estudio del estado del arte

Inicialmente, en el capítulo 2 realizamos una revisión crítica del estado del arte de la generación procedural de contenido para videojuegos y aplicaciones gráficas, enumerando las técnicas más utilizadas para este cometido y analizamos los diferentes acercamientos existentes, tanto en la literatura científica como en la industria.

Capítulo 3: Evaluación de resultados

En este capítulo proponemos un conjunto de métricas para medir la calidad de un generador procedural, así como los requisitos mínimos que un generador de estas características debe cumplir. Para ello analizamos los requisitos y aspectos deseables para generadores procedurales presentes tanto en el ámbito científico como en el industrial.

Capítulo 4: Representación volumétrica del terreno

Antes de proponer el método de generación en sí mismo, en este capítulo presentamos el sistema de representación de datos volumétricos en el que se apoya dicho método. En él, describimos el modo en que se representan los datos volumétricos en memoria, la manera mediante la que se accede a ellos y la composición de dichos datos. Además, explicamos una serie de conceptos periféricos que si bien no tienen incidencia directa en nuestra investigación, se ven influenciados por este sistema de representación, como el modo de renderizado de volúmenes o el sistema de almacenaje de los datos en disco. Finalmente, en este capítulo mostramos los resultados de una serie de experimentos que evalúan el tamaño de los terrenos utilizando este sistema de representación.

Capítulo 5: Generación del terreno volumétrico

En este capítulo introducimos el algoritmo de generación del terreno volumétrico, enumerando sus características principales, describiendo el algoritmo en sí mismo así como sus datos de entrada, y presentando los resultados obtenidos en una serie de experimentos que evalúan la velocidad con la que se ejecuta el método. Finalmente, y a modo de comparación visual, presentamos una serie de cortes de terreno reales que contienen diferentes estructuras y analizamos la capacidad de nuestro método para reproducirlas.

Capítulo 6: Generación de elementos jugables

En este capítulo presentamos los métodos propuestos por esta investigación para la generación de la capa jugable del mundo virtual, compuesta por dos tipos diferentes de estructuras: cuevas naturales y mazmorras manufacturadas. Inicialmente, presentamos una comparación entre estos dos elementos jugables, justificando esta división y describiendo las características de cada uno de ellos. A continuación, por cada uno de estos dos elementos, proponemos un método de generación, así como los resultados obtenidos en los diferentes experimentos para evaluar su velocidad de ejecución.

Capítulo 7: Conclusión

Para finalizar, en el capítulo 7 extraemos las conclusiones de la investigación llevada a cabo. Para ello, analizamos los resultados presentados en los capítulos anteriores y los comparamos con los requisitos iniciales utilizando las métricas propuestas. Además, analizamos la validación de la hipótesis inicial expuesta en la sección 1.4 utilizando

para ellos los resultados finales. Finalmente, enumeramos los puntos fuertes y débiles de nuestro método y proponemos una serie de posibles líneas de trabajo que permitan, en un futuro, extender esta investigación.

Apéndice: Prototipo

De manera adicional, se incluye a modo de apéndice una descripción del prototipo producido en esta investigación para realizar todos los experimentos descritos en los capítulos anteriores. Este prototipo implementa todos los métodos propuestos en esta investigación.

«¡Es peligroso ir solo! Toma esto.»

Old man (*The Legend of Zelda*, Nintendo, 1986)

CAPÍTULO

2

Estudio del estado del arte

La generación procedural de contenido es un campo que desde su nacimiento ha sido ampliamente estudiado. A pesar de ser un área que ha comenzado a explorarse recientemente, ha generado una gran cantidad de trabajos, incluidos varios artículos de revisión [HMVDVI11, TYSB11, RZL12, SdKT+09]. Además y a causa de las implicaciones comerciales que tiene en la industria de la informática gráfica, existen varias aplicaciones comerciales para la generación supervisada, así como una notable cantidad de videojuegos que utilizan contenido procedural profesionalmente.

Debido a la variación entre todos los tipos de elementos requeridos por los videojuegos, no existe un método de generación procedural de contenido que sea válido para todos ellos. En su lugar, cada investigación se centra en una serie de aspectos de uno o varios elementos diferentes, lo que hace que el criterio utilizado para clasificar estos métodos esté basado en el tipo de contenido que generan.

En esta sección realizamos una revisión crítica de los métodos existentes para la generación procedural, poniendo especial atención en aquellos métodos cuyo objetivo sea la generación de terrenos y mundos virtuales, así como sus elementos jugables. A pesar de ello, y aunque quede fuera del alcance de la investigación de esta tesis, también vamos a analizar el estado de la técnica de métodos que generan otro tipo de contenido relacionado con los mundos virtuales (por ejemplo, ciudades, edificios o plantas), ya que la filosofía detrás de dichas investigaciones es, a menudo, la misma.

Los objetivos de este análisis son los siguientes:

- Analizar y debatir los métodos tradicionales para la generación procedural, contextualizando el método propuesto en esta tesis y viendo cómo encaja en el estado de la técnica actual.
- Comprender las necesidades y la dirección que históricamente ha tomado esta área, tanto en el ámbito académico como en el industrial.

La estructura del capítulo está organizada de la siguiente manera: en la sección 2.1 realizamos una enumeración de las técnicas más utilizadas en la generación procedural; a continuación analizamos el estado actual de la técnica en la generación de mundos virtuales tanto en la literatura (sección 2.2), como en la industria (sección 2.3); y finalmente, en la sección 2.4 realizamos una breve conclusión sobre el estado del arte y del encaje de nuestra investigación en él.

2.1 Técnicas utilizadas para la generación procedural

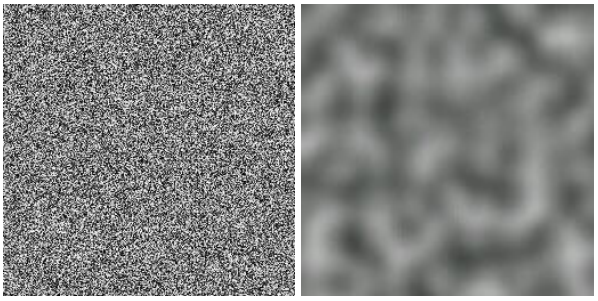
Como preámbulo al estado del arte explicamos brevemente aquellas técnicas que se utilizan en la generación procedural.

Como hemos dicho anteriormente, la generación procedural para juegos es un campo con aproximadamente 30 años de vida. Aunque este periodo parece un periodo corto en el tiempo, la evolución que ha sufrido la informática a nivel de *hardware* y *software* durante estos años ha hecho que aparezcan multitud de técnicas y métodos para la generación procedural de contenido. Además, el crecimiento del número de estas técnicas se ha visto favorecido por la cantidad de tipos diferentes de contenido que es necesario en un videojuego. Para enumerarlas y analizarlas, y debido a su número, varianza y complejidad, vamos a clasificarlas mediante los criterios utilizados por Hendrikx *et al.* [HMVDVI11].

2.1.1 Generadores de números pseudoaleatorios

Un generador pseudoaleatorio de números es un algoritmo que genera series de valores numéricos que simulan aleatoriedad (ruido blanco). Los valores obtenidos con los mejores generadores de números pseudoaleatorios son estadísticamente indistinguibles de una serie de valores aleatorios reales. Estos algoritmos se utilizan de manera exhaustiva por los generadores de contenido procedural.

Un caso particular de generadores de números aleatorios es el generador de ruido Perlin [Per85]. Los valores obtenidos con este generador no intentan simular aleatoriedad, sino que generan ruido mediante la interpolación entre unos pocos valores situados en el espacio, consiguiendo sensación de continuidad. Debido a esta característica, los valores obtenidos con este generador son perfectos para simular diferentes aspectos de la naturaleza, causa por la cual es uno de los algoritmos más utilizados en esta área.



(a) (b)

Figura 2.1: Comparación entre ruido blanco(a) y ruido Perlin(b).

La figura 2.1 muestra un ejemplo de un generador de números pseudoaleatorios tradicional (a) y de un generador de ruido Perlin(b). Para facilitar la comparación visual de los resultados de los dos generadores, los valores obtenidos están almacenados en una textura bidimensional donde el valor más bajo está representado por el color negro y el valor más alto por el color blanco (habitualmente estos valores son 0 y 1 respectivamente).

2.1.2 Gramáticas generativas

El concepto de gramáticas generativas pertenece al área que estudia la sintaxis de la lengua. Una gramática generativa proporciona un conjunto de reglas o principios que predicen correctamente las combinaciones que aparecen en oraciones gramaticalmente correctas para una determinada lengua. En el campo de la generación procedural se utilizan para operar sobre elementos codificados como palabras.

2.1.2.1 Sistemas de Lindenmayer

Estos sistemas [Lin68], también conocido como sistemas-L, utilizan una serie de operadores que describen las características de un objeto. Tradicionalmente, se han utilizado

para modelar el crecimiento de las plantas, generando estructuras similares a los fractales.

2.1.2.2 **Gramáticas de división**

Fueron introducidas por Wonka *et al.* [WWSR03] y tienen un comportamiento similar al de los sistemas-L. Estas gramáticas operan sobre una serie de formas mediante un conjunto de reglas que las subdividen y generan nuevas formas. Esta técnica se ha utilizado principalmente para generar modelos arquitectónicos de manera automática.

2.1.2.3 **Gramáticas de fachada**

Estas gramáticas fueron propuestas para la generación procedural por Larive y Gaildrat [LG06] y son similares a las gramáticas de división, al igual que las propuestas por Wonka *et al.*[WWSR03], se han utilizado principalmente para generar arquitectura, aunque sus resultados son más variados, llegando incluso a construir estructuras tan complejas como balcones.

2.1.2.4 **Gramáticas de forma**

Este tipo de gramáticas son similares a las dos anteriores, pero permiten generar estructuras mucho más complejas [SG71]. Al igual que el resto de gramáticas, también se han utilizado principalmente para generar modelos arquitectónicos. Fueron introducidas por Müller *et al.* [MWH$^+$06] y su funcionamiento se basa en el reemplazo de símbolos teniendo en cuenta el símbolo a reemplazar y sus vecinos.

2.1.3 **Procesamiento de imágenes**

Es el proceso mediante el cual se convierte una imagen de entrada en otra más adecuada para la aplicación objetivo. Estas técnicas tienen muchas utilidades dentro de la generación procedural, siendo la modificación de mapas de alturas y la generación de texturas procedurales sus dos máximos exponentes.

2.1.3.1 Segmentación y transformación binaria

El proceso de segmentación binario de una imagen da como resultado otra imagen basada en la inicial, pero con los posibles valores para cada píxel reducidos únicamente a dos (blanco o negro). El método más común para realizar esta conversión es mediante un umbral que determine cuándo un píxel pasa a ser de un valor o del otro. Una vez obtenida la imagen segmentada, se le pueden realizar diferentes operaciones de transformación como dilatar (añadir píxeles al borde de ciertas zonas), erosionar (quitar píxeles del borde de ciertas zonas), sumar imágenes o restarlas.

2.1.3.2 Filtros de convolución

Una convolución es un operador matemático que transforma dos funciones en una tercera modificando una de ellas mediante la otra. En procesado de imagen se utilizan habitualmente, con aplicaciones tan variadas como eliminación del ruido, suavizado o detección de bordes. En el campo de la generación procedural se utiliza principalmente para transformar texturas.

2.1.4 Algoritmos espaciales

Los algoritmos espaciales son una familia de algoritmos que modifican un espacio estructurado para generar contenido procedural.

2.1.4.1 *Tiling* y mapeado por capas

El *tiling* es una técnica en la cual el espacio se estructura en una rejilla cuyas celdas pueden tener diferentes formas (aunque normalmente son cuadradas o hexagonales), sobre las cuales se posicionan diferentes casillas (comúnmente llamadas *tiles*). De esta manera combinando los *tiles* de diferente forma se consigue crear superficies diferentes.

El mapeado por capas (evolución del *tiling*) es una técnica que combina diferentes *tiles* con zonas transparentes en cada celda de la rejilla, permitiendo crear efectos más complejos que con un *tiling* simple y obtener resultados mucho más variados.

Estas dos técnicas son muy utilizadas para la generación de terrenos en juegos en dos dimensiones. Además, el mapeado por capas permite a los juegos en dos dimensiones utilizar características reservadas a los juegos en tres dimensiones, como la inserción de elementos a diferentes profundidades de la escena.

2.1.4.2 **Subdivisión de rejilla**

Es el proceso mediante el cual se divide un objeto en una rejilla regular que, a su vez, se subdivide recursivamente mediante una aleatoriedad controlada, aumentando progresivamente su nivel de detalle. Fournier *et al.* [FFC82] fueron uno de los primeros en utilizar esta técnica para representar y renderizar terrenos.

Debido a la recursividad de esta técnica, a los terrenos generados por ella se les ha denominado comúnmente terrenos fractales.

2.1.4.3 **Fractales**

Una forma fractal es una figura formada por versiones más pequeñas de sí misma recursivamente. De esta manera, los objetos generados mediante fractales pueden llegar a tener el nivel de detalle que sea necesario sin ningún límite teórico. En el campo de la generación procedural tienen muchas aplicaciones, como la generación de plantas y árboles o la de terrenos estéticamente realistas.

2.1.4.4 **Diagramas de Voronoi**

Son diagramas que permiten dividir el espacio en un conjunto de sectores en base a una serie de puntos (llamados semillas) [Vor08]. El algoritmo asigna un sector a cada semilla, de tal manera que la semilla más cercana a cada punto de cada sector sea la semilla asignada al sector en el cual se encuentra. Estos diagramas se utilizan en multitud de ámbitos diferentes, como la indexación de ficheros, el movimiento de una entidad en un entorno [Aur91] o la generación de redes de carreteras [GMB06].

2.1.5 **Simulación de sistemas complejos**

Debido a la complejidad de algunos fenómenos naturales en ocasiones es necesario recurrir a modelos o simulaciones complejas para generarlos.

2.1.5.1 **Autómatas celulares**

Un autómata celular es un modelo matemático para un sistema dinámico que evoluciona en pasos discretos basado en una rejilla, donde cada celda tiene su estado y está sujeta a una serie de reglas [Neu96]. Estas reglas definen cómo el estado de cada celda

y el de sus vecinas afectan el estado de la siguiente celda. Así, se generan resultados aparentemente aleatorios, aunque en realidad, periódicos y sujetos a patrones. Los autómatas celulares pueden ser usados para modelar numerosos sistemas físicos que se caractericen por un gran número de componentes homogéneos y que interactúen entre sí. Esta técnica se ha utilizado para muchos fines, como la simulación de sistemas biológicos [CD98].

2.1.5.2 **Campos tensoriales**

Este tipo de campos son aquellos que tienen un tensor (definido como un vector de N dimensiones) asociado a cada punto y pueden utilizarse para definir ciertas características del entorno como redes de carreteras o distribución de ciudades [CEW+08].

2.1.5.3 **Modelos basados en agentes**

Un modelo basado en agentes es un modelo computacional que simula acciones e interacciones de individuos autónomos (llamados agentes) dentro de un entorno. Los agentes pueden aprender durante el proceso, y actúan en cada momento a favor de la manera en que ellos perciben sus propios intereses.

2.1.6 **Inteligencia artificial**

La inteligencia artificial es un área de investigación muy amplia que intenta simular comportamientos inteligentes. Existen muchas técnicas diferentes dentro de la inteligencia artificial que se utilizan en la generación procedural, entre las cuales destacan las que vemos a continuación.

2.1.6.1 **Algoritmos evolutivos**

Los algoritmos evolutivos están diseñados para explorar espacios de soluciones muy amplios desde un punto de vista basado en la teoría de la evolución. Representan las posibles soluciones como individuos (cuyas características se codifican en cromosomas) y hacen avanzar las generaciones cruzándolos, mutándolos y eligiendo a los mejores. Estos algoritmos tienen multitud de aplicaciones [Gol89], como la generación de terreno basado en diferentes criterios [RZL12].

2.1.6.2 **Redes neuronales**

Este tipo de redes son paradigmas de aprendizaje automático basados en el funcionamiento del sistema nervioso biológico y se utilizan para reconocer patrones o para clasificar y estructurar datos [Hay94]. Este tipo de redes tiene neuronas interconectadas con líneas de entrada y de salida a las que se asigna un peso, de tal manera que cada neurona evalúa todas las entradas que recibe y determina su salida.

2.1.6.3 **Satisfacción de restricciones**

La satisfacción de restricciones es el proceso de búsqueda de una solución para una serie de restricciones que imponen ciertas condiciones a un conjunto de variables [RNC+96]. Una solución es, por tanto, el conjunto de variables que satisfacen dichas restricciones. Normalmente, este proceso se aborda mediante algoritmos de búsquedas.

2.2 **Generación procedural de mundos virtuales**

Los mundos virtuales utilizados en los videojuegos son, normalmente, espacios amplios poblados por diferentes elementos, como accidentes geográficos, asentamientos de civilizaciones, o vegetación variada. Estas características hacen que estos mundos virtuales sean uno de los elementos que más atención han recibido por parte de la generación procedural. En esta sección hablamos sobre la generación de dichos mundos, y debido a las características de la investigación realizada en esta tesis, nos centramos en la generación del terreno y de los aspectos jugables que lo rodean.

2.2.1 **Generación de terrenos**

Es común que los videojuegos utilicen terrenos para representar el mundo en el cual se desarrollan. Estos terrenos, normalmente, se han representado mediante matrices bidimensionales que contienen la altitud del terreno en cada punto representando diferentes accidentes geográficos. En ocasiones es difícil diferenciar entre un terreno y un nivel, ya que algunos juegos se desarrollan en entornos que mezclan los dos conceptos. Por ejemplo, algunos *First Person Shooters* se desarrollan en entornos con pocas alternativas porque son, en esencia, niveles clásicos.

Existen diferentes perspectivas a la generación de terrenos, que tradicionalmente han sido generados mediante métodos fractales. Sin embargo, en los últimos años se ha

creado una nueva corriente de generación de terrenos mediante algoritmos evolutivos.

2.2.1.1 Terrenos fractales

Como ya hemos mencionado, la generación de terrenos se ha realizado comúnmente mediante algoritmos basados, en mayor o menor medida, en construcciones fractales. El objetivo de este acercamiento es conseguir terrenos extremadamente realistas en el apartado visual, sin prestar demasiada atención al apartado jugable del mismo.

Entre los algoritmos más antiguos utilizados para la generación de terrenos está la subdivisión de rejilla, la cual genera recursivamente nuevos puntos en una rejilla regular basándose en las características de los puntos adyacentes. Miller [Mil86] describe algunas variantes de este método, en las cuales la elevación de cada nuevo punto se calcula mediante la media de las alturas de los puntos de sus esquinas, a la que añade un desplazamiento aleatorio, que se modifica en cada iteración en función de lo abrupto que se desee el terreno. Otras investigaciones en esta área son las basadas en generadores de ruido fractal, como el ruido Perlin [Per85]. Fournier *et al.* [FFC82] y Voss [Vos85] utilizan métodos basados en esta técnica.

Estos algoritmos iniciales han acaparado la atención investigadora durante muchos años y se han realizado muchos trabajos sobre ellos que añaden diferentes aspectos a los métodos originales. Hay dos aspectos principales sobre los cuales han girado dichas investigaciones: por un lado, la simulación de la erosión para aumentar el realismo de los terrenos y por otro, el aumento del control a la hora de generar los terrenos, ya que los algoritmos vistos hasta ahora no permiten una gran influencia del diseñador en el resultado final.

Simulación de la erosión

La simulación de la erosión sobre terrenos fractales permite aumentar el realismo visual de dichos terrenos. Existen varios tipos de erosión que se pueden simular, siendo la erosión térmica y la erosión fluvial los dos tipos principales. Por un lado, la erosión térmica disminuye el ángulo de inclinación del terreno distribuyendo iterativamente material desde las zonas superiores a las inferiores hasta llegar al ángulo de estabilidad para el tipo de material que esté simulando. Por otra parte, la erosión fluvial se puede simular calculando la cantidad de agua y material disuelto en cada celda mediante la pendiente del terreno en la superficie.

Musgrave *et al.* [Mus93] [MKM89] tratan tanto la erosión térmica como la fluvial y Olsen [Ols04] discute distintas optimizaciones en la velocidad de generación, manteniendo una calidad aceptable en el resultado. Benes y Frosbach [BF01] proponen un método de erosión que se aplica sobre un modelo del mundo más complejo que una simple superficie. Su modelo para representar el terreno consiste en una serie de listas que definen la profundidad y la densidad de los materiales en cada punto, y permite incluso sistemas de cuevas subterráneas. Esta investigación es una de las pocas que tiene en cuenta las características volumétricas del terreno, aunque solo lo hace para simular la erosión y no para generar el terreno en sí mismo.

Los métodos que simulan la erosión en este tipo de terrenos son, con frecuencia, métodos muy lentos y costosos de ejecutar, ya que necesitan iterar muchas veces sobre el mismo terreno para conseguir resultados satisfactorios. Las últimas investigaciones se están centrando en portar este proceso a la GPU para mejorar así su rendimiento. Algunas de las investigaciones más relevantes son las realizadas por Anh *et al.* [ASA07], Mei *et al.* [MDH07] y Krištof *et al.* [KBKŠ09] que simulan la erosión fluvial en la GPU.

Aumento del control del diseñador.

Como hemos dicho anteriormente, la generación de terreno mediante algoritmos fractales no permite que los diseñadores influyan en el resultado. Por ello, existe un gran número de publicaciones que intentan mitigar este problema. Estas investigaciones han optado por diversas soluciones.

Algunos métodos se han basado en aumentar los parámetros del algoritmo generador de terrenos aunque normalmente sufren el problema de que dichos parámetros no son amigables para el diseñador ni es fácil de intuir cómo afectan al resultado final. Kamal *et al.* [KU07] proponen un método especializado en generar terrenos con una sola montaña o un solo cráter, definidos por una serie de parámetros. Doran *et al.* [DP10] proponen un método basado en agentes que permite una gran personalización del terreno generado. Otros métodos han mejorado la interacción con el diseñador permitiendo a este que realice un boceto del terreno a generar. Gain *et al.* [GMS09] proponen una técnica en la cual el diseñador define el contorno de las elevaciones. Por el contrario, en el enfoque de Zou *et al.* [ZSTR07] el diseñador determina los «caminos» que siguen las elevaciones desde un punto de vista vertical. Por su parte, Hnaidi *et al.* [HGA$^+$10] introducen otro método similar a los dos anteriores. De Carpentier y Bidarra [dCB09] optan por una original solución en la que el diseñador puede «pintar» directamente sobre las alturas del terreno.

2.2.1.2 Algoritmos evolutivos

La utilización de algoritmos evolutivos para la generación procedural de terrenos es una técnica que ha empezado a usarse recientemente. Este tipo de algoritmos permiten a los diseñadores un mayor control sobre las características finales que debe cumplir el terreno obtenido, así como la inclusión de otros tipos de elementos, como entidades jugables. Esto permite generar terrenos que cumplan los requisitos de múltiples géneros. Algunos buenos ejemplos de esto son [TDNL07] [SYT10].

Actualmente existen seis corrientes en esta área y prácticamente todos los trabajos de este campo se agrupan en alguna de estas seis ramas, las cuales las identificamos por el nombre de sus creadores.

Ong *et al.*

La primera investigación sobre el uso de algoritmos evolutivos para la generación de terrenos fue la de Ong *et al.* [OSKL05]. Su trabajo ha culminado en un programa llamado *Terrainosaurus*, desarrollado por Saunders [Sau06].

Su propuesta se divide en dos fases. En la primera fase, obtiene las líneas exteriores del terreno distorsionando un boceto creado por el diseñador. En la segunda fase, genera el mapa de alturas utilizando como modelo una serie de mapas de alturas aportados por el diseñador. En el algoritmo genético se establecen una serie de puntos en los mapas y se aplican una serie de operadores que los modifican subiéndolos, bajándolos o rotándolos. Estos puntos junto a sus operadores constituyen el genotipo de cada individuo. Además, tienen un área de influencia, por lo que los modificadores también afectan a los puntos de alrededor. El cruce entre individuos se realiza mediante recombinación en un punto y su función de aptitud mide la similitud con los terrenos aportados por el diseñador.

El algoritmo *Terrainosaurus* es adecuado para generar familias de terrenos similares, lo que puede ser útil dependiendo del propósito para el cual queramos utilizar los terrenos. Uno de los principales problemas de este método es que es necesario disponer previamente de un conjunto de terrenos similares al que queremos construir.

Ashlock *et al.*

Ashlock *et al.* aplica algoritmos evolutivos a un Sistema-L para generar terrenos similares a los terrenos fractales [AGB05].

El Sistema-L se expande en dos dimensiones y cada símbolo se traduce en una celda, generando una matriz de alturas. A cada símbolo se le asigna un valor de desplazamiento en la gramática del Sistema-L de tal manera que, cuando se expande, la altura del nuevo punto se modifica con los valores de desplazamiento de los puntos colindantes. El genotipo de los individuos está formado por una serie de reglas y valores de desplazamiento para cada símbolo de la gramática. Este método utiliza cruce de dos puntos entre individuos y calcula su aptitud mediante comparación con un terreno ideal proporcionado por el diseñador.

El principal problema de este método, igual que en el de Ong [OSKL05], es que necesitamos un terreno similar al que queremos construir.

Walsh y Gade

En su método, Walsh y Gade [WG10] utilizan algoritmos evolutivos para ajustar automáticamente los parámetros de un generador de terrenos. Estos parámetros incluyen la escala, el factor de escarpado, el nivel del agua y características atmosféricos. Este método no genera un terreno completamente nuevo, sino que ajusta uno ya existente.

El genotipo de los individuos en su algoritmo genético está formado por 8 bits para cada parámetro, siendo cada mutación una alteración de uno solo de estos bits. Este algoritmo combina diferentes individuos mediante el cruce en un punto de la cadena de bits de uno de los parámetros de dichos individuos. La aptitud de cada individuo se determina manualmente mediante un método interactivo. Walsh y Gade han propuesto posteriormente una función de aptitud alternativa [WG11] que se ejecuta de manera automática, basada en el ratio entre el orden y la complejidad del terreno para determinar su calidad estética.

No obstante, este método resulta de poca utilidad práctica para su aplicación en videojuegos ya que se centra más en el aspecto artístico y visual del terreno que en su utilidad dentro de una aplicación interactiva.

Frade *et al.*

Frade *et al.* han creado un método llamado *GenTP* [FFC09], que se encarga de crear funciones de altura mediante algoritmos evolutivos, y en el que una función de altura es una ecuación que se aplica sobre todos los puntos de un mapa de alturas para crear otro mapa diferente.

El genotipo en este algoritmo es un árbol de operaciones y evoluciona mediante mutaciones que añaden, eliminan o sustituyen operadores, que pueden ser numéricos básicos, trigonométricos u otras funciones personalizadas. Las funciones terminales crean un mapa de alturas sobre el que van operando el resto de operadores del árbol. En los primeros trabajos de Frade *et al.* [FFC09], estas funciones terminales eran generadores de ruido estocásticos. Sin embargo, posteriormente estos algoritmos se han sustituido por generadores de valores con repetibilidad [FFC10a]. Aunque inicialmente se utilizó una función de aptitud interactiva, investigaciones posteriores, para generar terrenos útiles para juegos, la han sustituido por dos funciones: por un lado, la accesibilidad, que favorece los terrenos con zonas llanas interconectadas [FFC10a] y por otro, la distancia obstáculo-arista, que favorece a los terrenos con obstáculos para el jugador [FFC10b].

Este método ha sido aplicado por sus mismos creadores en un juego comercial [RFF10], aunque tiene como resultado con demasiada frecuencia terrenos excesivamente planos y predecibles.

Togelius *et al.*

La propuesta de Togelius *et al.* [TPY10] se basa en la utilización de un algoritmo evolutivo multiobjetivo (MOEA) para la generación procedural de terrenos para juegos de estrategia. De esta manera, los terrenos generados mediante este método incluyen también elementos jugables como la posición de las bases de los jugadores u objetivos del mapa.

El proceso de generación se basa en la utilización de un mapa de alturas plano en el que se levantan montañas y se construyen entre ellas crestas mediante curvas gaussianas. El genotipo de los individuos está formado por dos valores para la desviación de la distribución gaussiana, dos valores para la posición de las montañas y uno para determinar la elevación de estas. La función de aptitud tiene en cuenta diferentes aspectos jugables como la dispersión de las bases de los jugadores, el equilibrio en el acceso a los recursos, o la cantidad de caminos entre bases. Utiliza un cruce binario probabilístico en el cual cada hijo se parece mucho más a uno de los padres que al otro.

Este método es el que más se acerca a la generación de terrenos directamente jugables y, como muestra, Togelius *et al.* ya han utilizado dicho método para generar terrenos para un juego comercial [TPB$^+$10]. El principal problema de esta técnica es que al estar orientada a generar terrenos con zonas planas separadas mediante crestas montañosas, solamente es útil para ciertos géneros de videojuegos.

Raffe *et al.*

El método de Raffe *et al.* [RZL11] construye un terreno mediante parches de terreno más pequeños utilizando algoritmos evolutivos. El diseñador comienza aportando un conjunto de terrenos de ejemplo y el algoritmo los divide en parches regulares que recombina para generar nuevos individuos.

El genotipo está definido mediante una matriz bidimensional con los identificadores de cada porción de terreno. El cruce se realiza generando un individuo exactamente igual que uno de los padres y dando una probabilidad a cada porción de terreno para ser sustituida por esa misma porción de terreno en el otro padre. La mutación se hace asignando a cada porción una probabilidad de ser sustituida por otra porción aleatoria de terreno. El proceso es interactivo, ya que el diseñador debe seleccionar los padres y qué porciones del terreno en dichos padres quiere excluir del algoritmo.

Los resultados son similares a los terrenos de muchos juegos pero tiene dos problemas principales. El primero de ellos es que es necesario aportar terrenos de ejemplo y no siempre se dispone de ellos. El segundo es que los terrenos generados por este algoritmo están limitados por los terrenos aportados como ejemplo, siendo estos los que se dividen y se recombinan. De esta manera, es imposible que se generen accidentes geográficos que no estuvieran presentes en los terrenos de ejemplo.

2.2.2 Cuevas

Las cuevas son, normalmente, uno de los aspectos críticos en los videojuegos que utilizan terrenos volumétricos procedurales.

Las primeras investigaciones en esta área se realizaron por Boggus y Crawfis [BC09b, BC09a]. Su método se basa en la simulación de la erosión de la roca por acción del agua. En una primera aproximación, limitaban las cuevas a dos superficies, formando el suelo y el techo de la cueva, pero más adelante superaron esta limitación, consiguiendo una mayor variedad de formas [BC10].

Peytavie *et al.* [PGGM09] presentan un *framework* completo para generar terrenos basados en *voxels*, que entre otras cosas, simula la erosión tanto térmica como fluvial, generando cuevas y formaciones rocosas complejas.

Por otro lado, Johnson *et al.* [JYT10] proponen el uso de autómatas celulares para la generación de sistemas infinitos de cuevas en tiempo real. Este método genera sistemas muy complejos y variados, pero tiene el inconveniente de que solo es válido para generar

cuevas bidimensionales.

Juncheng *et al.* [CCZ11b] proponen un método para generar cuevas basadas en *voxels* mediante ruido tridimensional. Este método se mejoró añadiendo estalactitas y estalagmitas a las cuevas [CCZ11a]. Este tipo de construcciones también han sido exploradas en el trabajo de Tortelli y Walter [TW09], que generan estalactitas y estalagmitas simulando la erosión y transporte de material que produce el agua.

2.2.3 Generación de aspectos jugables

Cuando hablamos de elementos jugables nos referimos a aquellos elementos que poseen los videojuegos y que hacen referencia directamente a la jugabilidad de los mismos. Por ejemplo, las entidades con las que el jugador puede interactuar o los puzles que debe solucionar pertenecen a este tipo de contenido.

La generación procedural de este tipo de elementos supone un reto mayor que el de generar aspectos meramente visuales del mundo virtual, ya que una mala decisión en este aspecto puede arruinar la experiencia jugable del entorno. Debido a esto, tanto el área académica como la industrial no le han prestado demasiada atención, dejándolo casi siempre en manos del diseñador.

2.2.3.1 Niveles

El concepto de nivel es un concepto abstracto que cambia de significado dependiendo del tipo de videojuego al que nos estemos refiriendo. Por ejemplo, un nivel en un juego de plataformas tradicional como *Super Mario Bros.* [Nin85] es el conjunto de habitaciones horizontales que forman cada pantalla, mientras que en juegos más complejos como *Dungeons & Dragons: Eye of the Beholder* [Wes90] o *Diablo* [Bli96] es cada piso de una mazmorra. Por tanto, un nivel es un área específica que engloba parte del mundo virtual en el cual se desarrolla el videojuego, y que el jugador debe superar mediante un conjunto encadenado de acciones. Así, el concepto de nivel mezcla aspectos de generación de entornos físicos con aspectos jugables.

Una de los primeros enfoques para la generación procedural de niveles es el propuesto por Compton y Mateas [CM06], corriente que continúan Smith *et al.* [STWM09]. Estos métodos están basados en el ritmo, más o menos regular, que la estructura de un nivel de un videojuego de plataformas debe mantener. Además, Dormans [Dor10] crea, mediante gramáticas generativas, niveles más complejos añadiendo una capa de aventura sobre el nivel generado.

Los algoritmos genéticos también se han utilizado para generar este tipo de contenido. Pedersen *et al.* [PTY09] modifican una versión *open source* de *Super Mario Bros.* y permiten utilizar contenido generado proceduralmente y terrenos infinitos que se generan según se avanza por el nivel. En su investigación utilizan un algoritmo genético, que evalúa la calidad de los individuos determinando las sensaciones que produce el nivel mediante una red neuronal, entrenada a partir de encuestas a jugadores. Este método ha sido modificado por Shaker *et al.* [SYT10] para ofrecer niveles personalizados para cada jugador.

Por su parte, Sorensons y Pasquier [SP10] utilizan una representación genérica, que permite generar niveles para diferentes géneros de videojuegos. Su algoritmo se basa en un genético que mezcla diferentes elementos que puede contener un nivel, y se asegura mediante un problema de satisfacción de restricciones de que todos los elementos estén interconectados. Además, evalúa la dificultad de los niveles mediante una función, dependiendo del tipo de juego y de las características que añaden dificultad al mismo (por ejemplo, en un videojuego de plataformas, la longitud de los saltos que hay que dar para superarlo).

Finalmente, Jennings-Teats *et al.* [JTSWF10] utilizan un algoritmo evolutivo para ir generando el nivel mientras el jugador va avanzando. Al igual que Compton y Mateas [CM06] y Smith *et al.* [STWM09], crea el nivel basándose en el ritmo, y va adaptando su dificultad al propio jugador.

2.2.3.2 **Entidades y su comportamiento**

Las entidades de los videojuegos son elementos habitualmente interactivos que pueblan el mundo virtual del juego. Estas entidades pueden ser enemigos que reaccionan a las acciones del jugador, simples viandantes que nada tienen que ver con la historia y caminan evitando colisionar contra otros elementos, una bomba que el jugador debe desactivar o, simplificándolo al máximo, una puerta que el jugador puede abrir o cerrar.

Mateas y Stern [MS05a] desarrollan *Façade* [MS05b], un entorno donde dos personajes virtuales reaccionan a lo que el jugador hace y dice.

Respecto a la simulación de multitud de entidades existen muchas investigaciones diferentes, siendo la de Reynolds [Rey87] una de las más influyentes, que utiliza técnicas de simulación de sistemas complejos.

Desde otro punto de vista, el videojuego *Galactic Arms Race* [Evo10] gestiona el armamento del jugador como una entidad interactiva, ya que evoluciona a lo largo del

tiempo según las preferencias del jugador [HGS09].

2.2.3.3 **Puzles**

A menudo se considera a los puzles un género independiente de videojuegos entre los cuales estaría, por ejemplo, el popular *Tetris* [Páz84]. A pesar de ello, los puzles son un recurso recurrente en otros muchos tipos de videojuegos, como en *The Legend of Zelda* [Nin86] o en *Resident Evil* [Cap96].

Iosup [Ios11] introduce un algoritmo para generar puzles para juegos MMO (*Massively Multiplayer Online*) basados en rejillas que recorre todo el espacio de soluciones posible.

De la misma manera, los algoritmos evolutivos también se han utilizado para generar puzles proceduralmente. Ashlock [Ash10] genera, mediante un algoritmo genético, laberintos de ajedrez y puzles cromáticos, siendo el genotipo de los individuos una matriz con la posición de las piezas (en el primer caso los colores y en el segundo las piezas de ajedrez) y la función de aptitud el número de movimientos necesarios para completar cada puzle.

2.2.4 **Generación de otros aspectos del mundo virtual**

Aunque la investigación de esta tesis se centra en la generación de terrenos y sus aspectos jugables, realizamos un análisis de los métodos existentes para la generación de otros aspectos de los mundos virtuales, como los cuerpos de agua, la vegetación o los entornos urbanos.

2.2.4.1 **Cuerpos de agua**

Los cuerpos de agua son elementos cuya disposición y forma está muy influenciada por el propio terreno. Por ello, algunos métodos se basan en un mapa de alturas ya generado, mientras que otros generan los cuerpos de agua a la vez que generan el terreno.

Kelley *et al.* [KMN88] crean una red de ríos generando un solo río y subdividiéndolo iterativamente. Después, a partir de dicha red, el método genera el terreno interpolando alturas e influenciando el proceso por el tipo de suelo y clima.

Prusinkiewicz y Hammel [PH93] utilizan un proceso fractal similar a la subdivisión

de rejilla (con la excepción de que utiliza triangulación en vez de una rejilla cuadriculada) para generar una red fluvial. Aquellos triángulos que no contienen río se procesan mediante un algoritmo de subdivisión de rejilla tradicional.

Belhadj y Audibert [BA05] utilizan un método más complejo que permite crear terrenos con crestas montañosas y redes fluviales. Comienzan generando una serie de crestas montañosas mediante una serie de curvas gaussianas. A continuación, determinan la red fluvial estableciendo su punto de partida en las montañas y calculando su ruta mediante física básica. El resto del terreno lo generan mediante técnicas fractales de subdivisión de rejilla.

Recientemente, Génevaux *et al.* [GGG+13] proponen un método que continúa la línea iniciada por Kelley *et al.* y que es capaz de crear ríos con diferentes características, como islas u orillas irregulares.

Por otro lado, la generación de lagos y costas no ha recibido el mismo grado de atención que la generación de ríos. Teoh [Teo08] realiza una primera aproximación, pero es necesario profundizar más es este tema para conseguir un mayor grado de realismo.

2.2.4.2 **Vegetación**

Debido a las características físicas de las plantas, la generación procedural de vegetación es un área de investigación clásica. Estas investigaciones se centran en dos aspectos diferenciados: por un lado, la generación de los modelos de las plantas en sí mismas, y por otro, la distribución de las plantas en un terreno.

Generación de árboles y plantas

La generación de modelos vegetales es la aplicación clásica de las gramáticas generativas y, más específicamente, de los sistemas de Lindenmayer (ver sección 2.1.2). Para ello, las cadenas de símbolos de dichas gramáticas se interpretan en 2D o en 3D determinando la estructura espacial de la planta [PL91] [PMKL01]. El principal problema de este acercamiento es que la aleatoriedad del resultado es muy alta, siendo muy complejo conseguir exactamente la planta que se desea, por lo que han surgido dos enfoques diferentes. Por un lado, el método de Štáva *et al.* [ŠBM+10] invierte el proceso de diseño de un sistema-L, permitiendo al diseñador especificar las características que desea en el resultado y generando después un sistema-L para crearlo. Por otro lado, el método de Beneš *et al.* [BŠMM11] introduce una técnica interactiva que permite dibujar la forma

de la planta a generar por el sistema-L.

Desde otro punto de vista, Lintermann y Deussen [LD99] proponen un método completamente diferente para la generación de plantas, en el que el diseñador determina las características de las plantas que desea mediante grafos formados por diferentes componentes de la planta, como puede ser una hoja, que a su vez está formada por el tallo y el limbo. Así, el sistema recorre el grafo generando el modelo final.

Recientemente, Pirk *et al.* [PSK+12] propone un método interactivo de modelado de árboles que reaccionan a las condiciones externas de un entorno dado.

Distribución de la vegetación

La distribución de la vegetación se ha abordado principalmente como la simulación de un ecosistema donde las plantas compiten entre ellas.

Deussen *et al.* [DHL+98] realizan una distribución desde un punto de vista competitivo. En su método, el diseñador debe definir las características de las diferentes especies de plantas, así como el mapa de alturas que define el terreno y el agua que contiene. Una vez establecidos estos parámetros, el algoritmo tiene en cuenta diferentes reglas para competir por el suelo, por la luz solar o por el agua, estableciendo en un proceso iterativo la posición de las plantas en el terreno. Los resultados de este método son muy realistas pero debido a la complejidad de sus cálculos es inviable su utilización para sistemas en tiempo real [SdKT+09].

Por otro lado, Hammes [Ham01] introduce un método que se ejecuta en tiempo real, y está basado en ecosistemas al igual que el de Deussen *et al.* [DHL+98] pero sin la simulación de la competición entre especies. En su investigación el diseñador determina qué tipo de ecosistema debe ir en cada zona del terreno y posiciona las plantas aleatoriamente en su interior. Para ello, utiliza información de alturas y su pendiente, así como algoritmos de ruido fractal.

Por último, Beneš *et al.* [BMJ+11] simulan la distribución de plantas en entornos urbanos. Su método tiene en cuenta tanto el plantado manual como la competición entre plantas por recursos.

2.2.4.3 Entornos urbanos

La generación de elementos propios de la civilización es un campo de investigación amplio dentro del área de la generación procedural, del cual existe una gran cantidad

de literatura. A su vez, existen muchas áreas menores que comprenden la generación procedural de diferentes elementos. A continuación, realizamos un análisis de las tres áreas más importantes dentro de este campo: la generación de redes de carreteras y caminos, la generación de edificios y la generación de ciudades uniendo las dos áreas anteriores.

Redes de carreteras

Las redes de carreteras y caminos son la clave de la jugabilidad de muchos videojuegos en los que la acción se desarrolla en un mundo abierto. Para generar redes de carreteras, se han utilizado técnicas dispares, tales como técnicas basadas en aplicación de patrones, gramáticas generativas, campos tensoriales o incluso simulación de sistemas basados en agentes. La principal dificultad a la que se enfrentan los métodos que intentan generar este tipo de contenido es la de encontrar un equilibrio entre aleatoriedad y estructura. Además, se debe cubrir el punto de vista del diseñador en otros muchos aspectos, como la interconexión de todos los puntos importantes del mapa, la obtención de la dificultad adecuada generando redes suficientemente complejas y el establecimiento de las carreteras más importantes para generarlas con otro tipo de características, como puede ser un mayor número de carriles.

Greuter *et al.* [GPSL03] proponen un método basado en la aplicación de patrones para generar redes de carreteras en el interior de ciudades, en el que se utiliza una matriz regular para simular la distribución de las calles de una ciudad moderna estadounidense similar a New York. Este sistema es demasiado simple y sus limitaciones son evidentes, ya que generan redes de carreteras poco realistas.

Otro método para generar redes de carreteras basándose en la aplicación de patrones es el propuesto por Sun *et al.* [SYBG02]. La base de este método es la aplicación sobre el terreno de uno de los patrones predefinidos y su posterior modificación dependiendo de ciertas restricciones locales. Sun *et al.* proponen un patrón basado en la densidad de población (creado a partir de diagramas de Voronoi [ver sección 2.1.4] de centros de población), un patrón radial (donde las carreteras se ordenan en diferentes capas circulares en torno a un punto central), un patrón de rejilla (donde las carreteras forman un patrón similar al propuesto por Greuter *et al.* [GPSL03]) y un patrón híbrido (que es una mezcla de los anteriores). El proceso de generación se basa en la aplicación directa de los patrones sobre el terreno, modificándolos dependiendo de las características de este. Los resultados de este método son menos artificiales que los del método propuesto por Greuter *et al.* [GPSL03], pero siguen sufriendo una notable falta de realismo.

Como ya hemos mencionado, las gramáticas generativas también se han utilizado para la generación de redes de carreteras. Parish y Müller [PM01] proponen un método, al que han llamado *CityEngine*, que utiliza un sistema-L extendido (llamado *Self-sensitive L-system*) para simular el crecimiento de una red de carreteras. Este método intenta interconectar todos aquellos centros de población con alta densidad siguiendo diferentes patrones, como pueden ser los utilizados por el método de Sun *et al.* Las parcelas que quedan entre carreteras se completan mediante un patrón que forma una rejilla regular.

Glass *et al.* [GMB06] también utilizan sistemas-L (ver sección 2.1.2) para la generación de redes de carreteras, pero los combinan con diagramas de Voronoi (ver sección 2.1.4). Además, validan el método mediante una serie de experimentos basándose en las calles existentes en algunos asentamientos informales de Sudáfrica, consiguiendo unos resultados bastante similares a los patrones observados.

Por su parte, Lechner *et al.* [LWW03] utilizan un sistema basado en agentes para este propósito, el cual divide la ciudad en diferentes tipos de zonas (residencial, comercial, gubernamental...). Este método utiliza dos agentes diferentes: el extendedor y el conector. El extendedor es el encargado de buscar zonas de la ciudad sin conexión entre ellas, mientras que el conector crea enlaces dentro de la red ya generada. Posteriormente, los autores han extendido su sistema con agentes que crean carreteras principales y otros que crean pequeñas calles secundarias [LRW$^+$06]. El mayor problema de este método es que necesita mucho tiempo para ejecutarse, aunque sus resultados son realistas, e incluso han llegado a utilizarse directamente en el videojuego *SimCity 3000* [Max99].

Chen *et al.* [CEW$^+$08] introducen un método para generar redes de carreteras mediante campos tensoriales (ver sección 2.1.5). En su acercamiento definen cómo crear diferentes patrones de carreteras sobre un campo tensorial, utilizando diferentes cantidades de ruido para generar carreteras de aspecto natural. Además, permiten al diseñador introducir sus propias carreteras mediante pinceles especiales. Los resultados de este método son visualmente realistas.

Galin *et al.* [GPMG10] afrontan el problema desde otro punto de vista. En su investigación utilizan el conocido algoritmo de búsqueda A* [HNR68] para establecer la posición de las carreteras, con una función de coste que determina la influencia de la inclinación, los cuerpos de agua o la vegetación en la trazada de las carreteras.

Así, podemos comprobar que la mayor parte de los métodos vistos anteriormente no permiten al diseñador influir en el resultado final. No obstante, Kelly y McCabe [KM07] introducen un editor llamado *CityGen* en el cual, para la generación de la red de carreteras, el usuario determina los nodos a interconectar y los patrones con los que

deben rellenarse las parcelas resultantes. Por otra parte, McCrae y Singh [MS09] hacen frente a este mismo problema con un método que crea una red de carreteras a partir de un boceto aportado por el diseñador.

Generación de edificios

Los edificios, junto a la red de carreteras, son el principal elemento que define la apariencia y configuración de cualquier entorno urbano. Además, en muchos videojuegos son uno de los aspectos clave que determinan la jugabilidad del mismo.

Greuter *et al.* [GPSL03], al igual que la solución que plantean para generar la red de carreteras de una ciudad, propone una solución simple pero efectiva para generar edificios, en la que se combinan una serie de polígonos primitivos y se extruyen a diferentes alturas, generando, de esta manera, edificios de oficinas modernos.

Sin embargo, la familia de métodos más utilizada para generar este tipo de contenido es la compuesta por las gramáticas generativas. Parish y Müller [PM01] utilizan un sistema-L aplicándolo sobre un rectángulo.

Wonka *et al.* [WWSR03] introducen el concepto de gramática de división (ver sección 2.1.2). Utilizando su método es posible generar edificios con diferentes estilos en cada piso; edificios realistas, pero con un grado de complejidad bajo.

Larive y Gaildrat [LG06] proponen las gramáticas de fachada, que como ya hemos mencionado en el apartado dedicado a dichas gramáticas, son muy similares a las de división. Los resultados con este método también son similares a los obtenidos con las gramáticas de división, pero con el añadido de que pueden generar estructuras más complejas.

Muller *et al.* [MWH$^+$06] proponen el uso de gramáticas de forma, basándose en los dos tipos de gramáticas anteriores. Este tipo de gramáticas se han utilizado en el campo de la arquitectura [KE81] y están especialmente bien provistas para la generación de fachadas. Este método comienza extruyendo un polígono del suelo que se divide en plantas, cuya fachada es, a continuación, dividida en paredes, ventanas y/o puertas. Finkenzeller *et al.* [Fin08] refinan este método generando edificios más consistentes.

Por otro lado, Müller *et al.* proponen un método completamente diferente para la generación de fachadas de edificios basadas en un ejemplo [MZWVG07], en el que el diseñador aporta una imagen de una fachada y, mediante filtros de tratamiento de imagen y gramáticas de forma, se genera automáticamente el modelo tridimensional del edificio.

Finalmente, y debido a la dificultad en la definición de este tipo de gramáticas han surgido una serie de corrientes para luchar contra ella. Lipp *et al.* [LWW08] proponen un método donde el diseñador puede ver el efecto de cada nueva regla que introduce en dicha gramática de manera interactiva.

Distribución de edificios

La distribución de edificios para generar entornos urbanos es, en esencia, la combinación de los métodos expuestos en los dos subapartados anteriores. Podemos comprobar incluso, que existen investigadores que tratan los dos puntos en el mismo trabajo, como Greuter *et al.* [GPSL03], que crean edificios limitados por las regiones de la rejilla que define la ciudad, o Parish y Müller [PM01], que dividen las secciones entre carreteras en parcelas y crean un edificio en cada parcela, obviando aquellas que no tienen acceso directo a la calle. Ya hemos mencionado también el editor *CityGen* (desarrollado por Kelly y McCabe [KM07]), que utiliza un método interactivo para generar carreteras secundarias y parcelar el terreno entre ellas.

Recientemente, Emilien *et al.* [EBP+12] propone un método que genera pequeños asentamientos rurales sobre un terreno dado. Este método, comienza generando una red de carreteras y, dependiendo del tipo de asentamiento que se quiere crear, va distribuyendo los edificios iterativamente con diferentes criterios.

Por otro lado, Vanegas *et al.* [VKW+12] proponen un método interactivo y comparan sus resultados con datos de ciudades reales, obteniendo parcelas muy realistas.

2.2.4.4 Interiores

La generación procedural de interiores es un problema muy complejo y directamente subordinado al uso que se quiera dar a dicho interior. No es lo mismo generar el interior de una pequeña vivienda unifamiliar, el de una planta de oficinas o el de una mazmorra de la edad media. Para llevar a cabo la generación de este tipo de contenido existen dos líneas diferentes de investigación. Por un lado, está la distribución de las diferentes estancias dentro de las paredes principales del edificio, y por otro, la disposición de los muebles y otros accesorios en cada habitación.

División en estancias

El problema de la división de estancias en un plano es un problema de distribución espacial [Lig00]. Algunas soluciones iniciales intentan tener en cuenta todas las posibles distribuciones para un número limitado de habitaciones [Gal81], pero debido al alto número de posibilidades resulta imposible llevarlo a la práctica. Schwarz *et al.* [SBS94] proponen un método en el cual el arquitecto define las habitaciones, su uso y su conectividad. Por su parte, Boyd y Vandenberghe [BV04] proponen un método similar, con la excepción de que la distribución inicial debe realizarse manualmente.

Rau-Chaplin *et al.* [RCMLS96] demuestran que es posible generar planos de planta mediante gramáticas de forma. En su investigación crean un esquema de alto nivel con las habitaciones básicas, que se agrupan en zonas funcionales (espacios públicos, privados, o semiprivados) y a las que posteriormente se les asigna su rol específico. El principal problema de este método es que no está diseñado para generar contenido con un grado mayor de detalle.

Michalek *et al.* [MCP02] generan la distribución de espacios de apartamentos rectangulares mediante algoritmos genéticos utilizando la conectividad entre habitaciones.

Desde otro punto de vista, Hahn *et al.* [HBW06] presentan un método para la generación en tiempo real de la distribución interna de edificios de oficinas, para lo que utilizan un sistema de subdivisión de espacios (en plantas, pasillos y habitaciones) que depende de la posición del observador, y permite incluso descartar habitaciones ya generadas para volver a crearlas después exactamente igual. Marson y Musse [MM10], por su parte, introducen un método similar de división de habitaciones basado en el algoritmo *Squarified Treemaps* [BHVW00]. Su método comienza con el contorno del edificio y un conjunto de habitaciones definidas por su funcionalidad y área deseada. El algoritmo subdivide el espacio en función de dichas habitaciones, generando incluso pasillos para interconectarlas.

Martin [Mar06] introduce un método en el que primero se crean las habitaciones y después, a partir de estas, se define el contorno final del edificio. Mediante una gramática definida por el usuario se genera un grafo con las habitaciones y sus interconexiones. Para generar después el espacio físico que ocupan estas habitaciones, utiliza un método de expansión basado en la presión. También Tutenel *et al.* [TBSdK09] utilizan un método de expansión para determinar el espacio que ocupan las habitaciones, pero difiere en su posicionamiento. Este método utiliza una capa semántica que relaciona las habitaciones entre sí y les aplica diferentes restricciones basadas en su rol. Otro método que utiliza este tipo de técnicas es el propuesto por Lopes *et al.* [LTS+10], en el que

se parte del contorno vacío del edificio, donde se posicionan las semillas iniciales de cada habitación mediante criterios de adyacencia, y donde finalmente, se extienden las habitaciones mediante presión.

Por último, Merrell *et al.* [MSK10] proponen un método para generar edificios mediante una red bayesiana entrenada con datos del mundo real. Los resultados obtenidos con este método son visualmente realistas y variados, aunque está limitado exclusivamente a la generación de edificios residenciales unifamiliares.

Distribución de muebles y accesorios

Una de las ideas sobre las que se basan las investigaciones en esta área es la de la asociación de objetos, introducida por Bukowski y Séquin [BS95]. Este método está diseñado para manipular objetos dentro de un entorno manteniendo la coherencia entre ellos.

Más adelante, Kjølaas [Kjø00] presenta un método para generar la distribución de una habitación basado en plantillas que sufren mutaciones, pero que tiene la desventaja de que solamente es válido para habitaciones rectangulares.

Muchas investigaciones han recurrido a métodos de inteligencia artificial para solucionar este problema. Sánchez *et al.* [SLRLG03], por ejemplo, utilizan un algoritmo genético para este propósito, cuyo genoma está basado en un árbol de operaciones entre diferentes elementos de la habitación. Larive *et al.* [LLRG04] presentan un método de búsqueda basado en ciertas restricciones que el diseñador debe asignar a los objetos y a sus interrelaciones.

Por su parte, Germer y Schwarz [GS09] proponen la utilización de un sistema basado en agentes, cuyos resultados son visualmente realistas, sin embargo, necesita una entrada de datos muy especializada y directamente relacionada con el método en vez de con el dominio del problema.

Más recientemente, Yu *et al.* [YYT+11] introducen un método que recibe como entrada una serie de organizaciones positivas de los elementos. En su enfoque el primer paso es la extracción de características de los ejemplos otorgados. A continuación la habitación se organiza aleatoriamente, y seguido, se optimiza mediante un algoritmo de búsqueda. Dicha búsqueda se limita minimizando una función de coste basada en aspectos como la accesibilidad o la visibilidad de los elementos.

2.3 La generación procedural en la industria

La industria de los videojuegos ha adoptado técnicas de generación procedural de contenido desde su descubrimiento. En esta sección hacemos un repaso del *software* más importante que utiliza la generación procedural. Debido a que muchos juegos y programas utilizan estas técnicas para generar elementos que quedan fuera del alcance de esta tesis, vamos a centrarnos en aquel *software* que utilice la generación procedural para generar mundos virtuales y sus aspectos jugables.

2.3.1 Herramientas comerciales que hacen uso de la generación procedural

Existen muchas herramientas diferentes para generar mundos virtuales, cada una centrada en unas pocas características del terreno. A pesar de ello, ninguna de estas soluciones tiene en cuenta la generación de la capa jugable que los mundos virtuales deberían tener para ser útil en una aplicación interactiva.

Las principales herramientas en esta área son aquellas que tienen como objetivo la generación del terreno del mundo virtual. Dentro de este grupo hemos seleccionado las soluciones más conocidas y utilizadas: *TerraGen* [Pla09], *L3DT* [Bun03], *World Machine* [Wor05] y *GeoControl* [Ros06].

La generación de terrenos utilizada por *Terragen* [Pla09] se basa en una red de nodos, en la que cada nodo representa una operación, como añadir color o añadir ruido al terreno. Los resultados de esta herramienta son sorprendentemente realistas y, por ello, se han utilizado en películas y series de televisión.[1] Sin embargo, el principal problema es que requiere mucha experiencia y conocimientos técnicos para obtener los resultados deseados.

La herramienta *L3DT* [Bun03] permite diseñar el terreno mediante la edición de una versión en baja resolución. Este mapa se edita mediante pinceles especiales y se puede especificar tanto la forma general de terreno como la fuerza y la posición de ciertos efectos, pasando por la erosión o erupciones volcánicas. Para generar el terreno final cada celda de este mapa en baja resolución, se expande a una sección del terreno de 64×64 puntos, aplicando todos los modificadores necesarios para ello, como el ruido o la erosión. La principal ventaja de esta herramienta es que la interacción con el diseñador es muy simple y no requiere grandes conocimientos específicos.

[1] http://planetside.co.uk/galleries/tg-in-film

World Machine [Wor05] es una herramienta para generar terrenos en la que el diseñador se limita a determinar los pasos para generar dicho terreno y el orden en el cual se ejecutan. Los pasos tienen un dato de entrada, un dato de salida y varias entradas de parámetros. Estos pasos pueden ser generadores (generan el terreno básico), modificadores (modifican la entrada mediante diferentes algoritmos) o salidas (guardan el terreno en un formato específico). El control que permite esta herramienta sobre el resultado obtenido es elevado.

GeoControl [Ros06] genera terrenos mediante técnicas fractales de subdivisión de rejilla. Permite al diseñador definir líneas en el mapa sobre las cuales se crean cordilleras montañosas, y además aplicar diferentes modificadores a dicho terreno, como el suavizado o la simulación de la erosión. Los resultados son, en general, montañas visualmente realistas, aunque para generarlas se requieren conocimientos específicos.

La generación procedural de ciudades también es un campo lo suficientemente extendido como para que existan soluciones comerciales con este propósito. Algunas de ellas son: *CityEngine* [Esr08] es una herramienta para generar entornos urbanos basados en gramáticas de forma. *Ürban PAD* [Gam] es una solución similar basada en nodos, que ya no tiene soporte debido a que la empresa creadora ha desaparecido. *CityScape* [Pix] mezcla métodos manuales e interactivos, pero al igual que *Ürban PAD* [Gam] ha dejado de recibir soporte.

Las generación de árboles y plantas también es un campo en el cual la industria a puesto sus esfuerzos. *SpeedTree* [Int09] está especialmente diseñado para generar grandes cantidades de vegetación, y ha sido utilizado en varias películas y videojuegos. *XFrog* [Gre96] está basado en la investigación de Lintermann y Deussen [LD99] y es directamente compatible con los terrenos de *TerraGen*. *Onyx* [Ony92] es una herramienta menos conocida, pero sus resultados son muy detallados y realistas.

2.3.2 **Videojuegos con mundos virtuales procedurales**

Además de las herramientas que ya hemos visto, existe un gran conjunto de videojuegos que utilizan entornos virtuales procedurales. En este apartado recopilamos los más importantes y analizamos qué y cómo lo generan.

A diferencia de las herramientas *software* para generar mundos virtuales procedurales, existen juegos que generan contenido procedural directamente jugable. Esto es debido a que este tipo de contenido tiene una fuerte dependencia del videojuego para el cual se genera, y las herramientas vistas en el apartado anterior intentan ser útiles

para todo tipo de objetivos.

La mayor parte de los videojuegos que utilizan contenido procedural, lo hacen exclusivamente para la generación del mundo virtual.

En *The Elder Scrolls II: Daggerfall* [Bet96] se utilizaron técnicas de generación procedural supervisada para generar el mundo virtual. Gracias a ello este mundo llegó a doblar en extensión a Gran Bretaña. Otros casos similares, aunque la extensión de sus mundos virtuales no alcanza la del anterior, son *Just Cause* [Ava06] o *FUEL* [Aso09]. Ninguno de estos juegos utiliza terrenos volumétricos, limitando su representación a mapas de alturas convencionales.

El juego que popularizó los terrenos volumétricos es *Minecraft* [Moj11]. En él, el terreno se genera de manera *online*, utilizando entre otras técnicas, combinaciones de ruido Perlin [Per85].

Terraria [Re-11] es la versión bidimensional de *Minecraft*. En este juego se genera un terreno volumétrico, pero con la excepción de que se realiza de manera híbrida entre *offline* y *online*, es decir, el terreno se crea sin intervención del diseñador y en tiempo de ejecución, pero se realiza en un paso anterior a la sesión de juego en sí misma. El terreno de *Terraria* también está basado en capas de materiales simples.

Por otro lado, existe una serie de videojuegos en los cuales absolutamente todo su contenido se genera mediante técnicas procedurales.

Ya hemos hablado de *Elite* [BB84], juego de simulación espacial en el cual hay ocho galaxias con 256 planetas cada una. Todos los aspectos de estos planetas (posición en el espacio, precios de las mercancías en sus mercados particulares, su nombre y los detalles locales) se generan proceduralmente utilizando generadores de números aleatorios.

Slaves to Armok: God of Blood Chapter II: Dwarf Fortress[1] [Bay06] es un videojuego de gestión de recursos que se desarrolla en un mundo procedural. Este mundo comienza con un terreno fractal que se complementa con características medioambientales. Posteriormente el sistema genera la historia de dicho mapa, creando diferentes asentamientos y simulando su desarrollo haciendo pasar los años. El resultado final es un mundo virtual completo y robusto, tanto en su estructura física como en su historia.

.kkrieger [.th04] es un videojuego de acción en primera persona cuyo contenido total (mundo virtual, enemigos, armas, texturas, efectos especiales, etc.) es generado mediante técnicas procedurales, por ello, solo ocupa 97.280 *bytes* en disco.

[1] Conocido comúnmente como *Dwarf Fortress*.

Otro juego que utiliza técnicas procedurales de manera exhaustiva es *Spore* [Max08]; presenta una galaxia donde todo es procedural, desde los planetas hasta las criaturas que viven en ellos y sus ciudades.

Recientemente ha aparecido en el mercado *Starbound* [Chu13], juego similar a *Terraria* que basa todo su desarrollo en los generadores de contenido procedural. En él es posible explorar una galaxia enorme donde todo está generado proceduralmente: los planetas (422,22 cuatrillones de planetas explorables a pie), las armas (más de dos millones de armas diferentes), los enemigos (un número incalculable de enemigos diferentes según los desarrolladores) o las mazmorras de cada planeta.

2.4 Conclusión y encaje de esta tesis en el estado del arte

Como hemos visto en este capítulo, la generación procedural de mundos virtuales es un área madura en el ámbito industrial además de en el académico. Existen métodos para la generación de múltiples aspectos de los mundos virtuales con el suficiente éxito para que sean directamente utilizados y explotados profesionalmente. A pesar de ello, la carencia de investigaciones en la generación de terrenos volumétricos ha ocasionado que los algoritmos de generación de este tipo de terrenos que utiliza la industria no estén publicados ni validados por la comunidad científica. Asimismo, uno de los daños colaterales de la poca investigación realizada en la generación de terrenos volumétricos es que no existen métodos de generación de contenido jugable específico para este tipo de terrenos, y que los existentes para mundos virtuales convencionales no han sido adaptados ni evaluados en terrenos volumétricos.

Además, la falta de consenso, métodos y criterios estandarizados y extendidos para evaluar los generadores procedurales causa que cada publicación, en el mejor de los casos, utilice sus propios criterios de evaluación, y en el peor, no evalúe de manera alguna.

En este contexto se desarrolla la investigación de esta tesis, con el objetivo de paliar estas carencias encontradas en el estado del arte de la generación procedural de mundos virtuales.

«¡No dispare, soy del equipo de ciencia!»

Científico (*Half Life*, Valve, 1999)

CAPÍTULO

3

Evaluación de los resultados

EVALUAR los resultados es un problema recurrente en todos los campos de la informática gráfica y la generación procedural. Debido a la naturaleza subjetiva de los resultados que obtienen los métodos de generación procedural, es muy difícil encontrar una colección de métricas de evaluación objetiva y que, a su vez, sea capaz de catalogar los métodos correctamente. Por un lado, algunos autores que basan su investigación en el realismo gráfico realizan este proceso mediante una comparación visual subjetiva con ejemplos reales [VKW+12]. Por otro lado, otros autores prefieren medir la complejidad de la escena y el tiempo que ha tardado su método en generarla [PM01], obviando la calidad subjetiva del resultado. Ninguno de estos métodos otorga resultados que representan realmente la calidad final del método que proponen.

Para superar este problema en esta investigación, sintetizamos aquellos aspectos de los generadores procedurales considerados como críticos en la literatura (enumerados brevemente en la sección 1) y en la industria, y proponemos una serie de métricas para evaluarlos de una manera más objetiva y global que las enumeradas en el párrafo anterior. Además, analizamos los niveles mínimos para dichos aspectos utilizados en ambos mundos (académico e industrial), proponiendo un conjunto de requisitos comunes para todos los generadores procedurales de mundos virtuales.

Lo que resta de capítulo se organiza de la siguiente manera: en la sección 3.1 definimos los aspectos críticos para un generador procedural, además, proponemos y deta-

llamos las métricas para cada uno de ellos; en la sección 3.2 analizamos los requisitos para cada uno de estos aspectos críticos (la sección 3.2.1 se enfoca hacia los requisitos presentes en la literatura, mientras que la sección 3.2.2 hacia los presentes en la industria); y en la sección 3.2.3 ponemos en común todo lo anterior y establecemos el conjunto de requisitos final.

3.1 Aspectos críticos y métricas de evaluación

Algunos de los aspectos críticos sobre los cuales se construyen las métricas de evaluación ya han sido nombrados en la sección 1. Estos aspectos son una síntesis de los requisitos propuestos por algunos autores [DP10, Sau06, OSKL05].

En esta tesis recopilamos, compatibilizamos y completamos este conjunto de aspectos críticos que se encuentran dispersos por la literatura. Además, proponemos una catalogación de estos aspectos en dos grupos bien diferenciados: orientados al **diseñador** (aquellos aspectos que influyen en el proceso de diseño y/o generación de un mundos virtual) y orientados al **usuario** (aquellos que condicionan la experiencia de usuario al interactuar con los mundos virtuales resultantes).

Con la finalidad de aumentar la usabilidad y legibilidad de estas métricas, les asignamos un valor numérico (indicado junto a cada nivel en la métrica). En apartados posteriores de esta tesis se hace referencia a dichas métricas utilizando este valor numérico.

3.1.1 Orientados al diseñador

Los aspectos críticos orientados al diseñador y sus métricas son los siguientes:

- **Innovación:** Capacidad del generador procedural de obtener elementos inesperados por el diseñador. Sus posibles valores son:

  0. Sin innovación. Los resultados son completamente predecibles, presentando los mismos elementos sin cambios entre ellos.

  1. Todos los resultados obtenidos presentan los mismos elementos, aunque con leves cambios en sus propiedades.

  2. Los resultados obtenidos presentan elementos inesperados, además de diferencias significativas entre elementos del mismo tipo.

- **Usabilidad:** Capacidad del generador procedural de ocultar los aspectos técnicos del método al diseñador. Sus posibles valores son:

  0. Los parámetros de entrada están relacionados con aspectos técnicos.

  1. Los parámetros de entrada están relacionados exclusivamente con el resultado, evitando todos los aspectos técnicos.

- **Control:** Nivel de personalización de los resultados que permite el generador procedural al diseñador. Sus posibles valores son:

  0. El método genera elementos en los cuales no se puede influir de ninguna manera.

  1. El diseñador no puede influir en todos los aspectos del resultado.

  2. El diseñador puede influir en todos los aspectos del resultado, pero no puede corregirlos de manera interactiva.

  3. El diseñador puede influir en todos los aspectos del resultado y puede corregirlos de manera interactiva si es necesario.

- **Ampliabilidad:** Capacidad del generador procedural para obtener mundos virtuales de diferentes tipos. Sus posibles valores son:

  0. Ninguna. El método genera todos los mundos virtuales del mismo tipo siempre.

  1. El método puede generar mundos virtuales de diferentes tipos, pero requiere cambios técnicos internos.

  2. El método puede generar mundos virtuales de diferentes tipos mediante cambios en los parámetros de entrada o en la interacción con el diseñador.

- **Escalabilidad:** Capacidad del generador de obtener mundos virtuales a diferentes escalas y nivel de detalle. Sus posibles valores son:

  0. El método genera todos los mundos virtuales con la misma escala.

  1. El método puede generar mundos virtuales a diferentes escalas.

3.1.2 **Orientados al usuario**

Los aspectos críticos orientados al usuario y sus métricas son los siguientes:

- **Estructura:** Capacidad del generador procedural de construir elementos reconocibles integrados entre sí. Sus posibles valores son:

  0. Los mundos virtuales obtenidos no presentan estructuras reconocibles; se asemejan al ruido aleatorio.

  1. Los mundos virtuales obtenidos contienen elementos reconocibles, pero no se encuentran integrados entre sí.

  2. Los mundos virtuales obtenidos presentan estructuras reconocibles que se integran entre ellas.

- **Interés:** Capacidad del generador procedural de obtener resultados interesantes con los que interactuar. Sus posibles valores son:

  0. Los mundos virtuales obtenidos no presentan estructuras explorables.

  1. El método genera mundos virtuales explorables.

  2. Los mundos virtuales obtenidos presentan estructuras explorables y, además, motivan dicha exploración de alguna manera.

- **Velocidad:** Velocidad del generador procedural en la obtención de los resultados. Sus posibles valores son:

  0. El resultado se genera durante el desarrollo, ya sea porque el método sea supervisado o porque requiere demasiado tiempo y/o recursos para ejecutarlo en la aplicación final. Método *offline*.

  1. El resultado se genera en tiempo de ejecución en un paso previo a la interacción con el jugador. Es decir, puede ejecutarse sin la supervisión del diseñador, pero no es lo suficientemente rápida para generarse de manera completamente *online*. Método híbrido entre *offline* y *online*.

  2. El resultado se genera mientras el jugador va explorándolo. El método requiere unos pocos milisegundos para ejecutarse, y permite extender un resultado ya creado. Método *online*.

- **Realismo:** Capacidad del generador de obtener mundos virtuales factibles. Sus posibles valores son:

  0. Los resultados son imposibles en cualquier realidad (por ejemplo, ruido aleatorio).

  1. Los resultados son imposibles en nuestra realidad, pero factible en otras realidades (por ejemplo, islas volantes).

2. Los resultados se basan en nuestra realidad, pero sin llegar a simular fenómenos naturales.
3. Los resultados son el producto de una simulación de nuestra realidad.

3.2 Nivel deseable cada aspecto crítico: requisitos

Las características que un método de generación procedural de mundos virtuales debe tener no son fáciles de definir. Dependiendo del propósito del generador o de la utilidad prevista para sus resultados, pueden variar en gran medida entre diferentes casos. En esta sección realizamos un proceso de análisis de los requisitos para este tipo de generadores presentes en la literatura, así como las particularidades de los generadores utilizados en la industria, con el fin de obtener unos requisitos comunes, genéricos y extendidos por la comunidad académica, a la vez que utilizados por la industria. Para realizar dicho análisis y poder determinar de una manera más exacta los niveles deseables para cada característica, hemos utilizado las métricas definidas en la sección 3.1 haciendo referencia a ellas por su nivel numérico.

3.2.1 Requisitos definidos por la literatura

A continuación analizamos las referencias que se hacen a las características previamente establecidas en las diferentes publicaciones, para obtener los requisitos que propone la comunidad científica para cada una de ellas. En algunas de estos trabajos no solamente se hace referencia a las características del método de generación en sí, sino que también se incluyen los requisitos del *framework* utilizado para la generación supervisada del mundo virtual. Debido a que este tipo de herramientas están fuera del alcance de la investigación de esta tesis, nuestro análisis obvia dichos requisitos.

3.2.1.1 Innovación

Ong *et al.* [OSKL05] definen el concepto de innovación en la generación de mundos virtuales como la capacidad de un generador de crear elementos originales e impredecibles que no deben estar, necesariamente, definidos en los datos de entrada. Las investigaciones de Ong *et al.* [OSKL05], Saunders [Sau06], y Doran y Parberry [DP10] también imponen esta característica a sus respectivos métodos, aunque simplemente dicen que la innovación debe de ser la mayor posible.

3.2.1.2 **Usabilidad**

La facilidad de uso de los métodos propuestos es uno de los requisito más extendidos en la literatura. En todos los trabajos que hacen referencia a las características deseables previamente identificadas ([OSKL05], [DP10], [STdKB08] y [Sau06]) se incluye la usabilidad. Según estos trabajos, todos los parámetros de entrada deben ser intuitivos y evitar conceptos técnicos.

3.2.1.3 **Control**

La capacidad del método para que el diseñador influya en el resultado es el otro requisito más extendido en la literatura. Las publicaciones que la nombran ([OSKL05], [DP10], [STdKB08] y [Sau06]) determinan que el diseñador debe ser capaz de influir en todos los aspectos del resultado, incluso de manera interactiva.

3.2.1.4 **Ampliabilidad**

La característica de ampliabilidad hace referencia a la capacidad que tiene un generador de mundos virtuales para generar diferentes tipos de mundos con pequeños cambios, ya sea solamente en los parámetros de entrada o en el propio método en sí. Este requisito se tiene en cuenta tanto por Ong *et al.* [OSKL05], como por Saunders [Sau06], y Doran y Parberry [DP10]. Todos ellos coinciden que para facilitar la adaptabilidad a diferentes propósitos de un generador de mundos virtuales debe permitir añadir nuevos tipos de terrenos sin requerir cambios internos en el método.

3.2.1.5 **Escalabilidad**

Los cuatro trabajos que recopilan las características deseables de un generador de mundos virtuales ([OSKL05], [DP10], [STdKB08] y [Sau06]) contemplan este requisito, sin embargo, son pocas las investigaciones que lo tienen en cuenta explícitamente. Así, atendiendo a aquellas publicaciones que consideran este aspecto como crítico, los resultados obtenidos con el generador deben estar convenientemente detallados independientemente de la escala.

3.2.1.6 **Estructura**

En el método de Smelik *et al.* [STdKB08] todos los elementos deben ser reconocibles y debe estar integrados entre sí sin crear estructuras aleatorias inesperadas. Además, tanto Saunders [Sau06] como Doran y Parberry [DP10] también tienen en cuenta este requisito.

3.2.1.7 **Interés**

Doran y Parberry [DP10] definen el interés de los mundos virtuales obtenidos por un generador procedural como la capacidad de creación de estructuras interesantes y/o explorables por los usuarios. El nivel que se debe cumplir en esta característica está directamente relacionado con el propósito del resultado, por lo que simplemente establecen que el resultado debe ser interesante para el usuario, sin entrar en más detalle.

3.2.1.8 **Velocidad**

La velocidad en la generación de mundos virtuales es uno de los requisitos más variables. En la mayor parte de los métodos presentes en la literatura es obviado, ya que se focalizan en la generación supervisada u *offline* [STdKB08]. Por su parte, Doran y Parberry [DP10] suponen que esta característica se cumple cuando los resultados están listos en el momento en el cual son requeridos para la interacción.

3.2.1.9 **Realismo**

Ninguno de los trabajos que recogen las características deseables para un generador de mundos virtuales hace una referencia explícita al realismo en sí como un requisito, aunque algunos de ellos lo incluyen al definir la estructura de los resultados ([Sau06], [DP10]). Sin embargo, todos ellos coinciden, ya sea explicita como implícitamente, en que el resultado debe simular la realidad.

3.2.1.10 **Conclusión**

Como podemos ver, los requisitos establecidos por la literatura intentan conseguir métodos de generación óptimos en todos los aspectos, excepto en el realismo y en aquellos que no encajan con el objetivo del método (interés, velocidad o usabilidad). Esto ocurre debido a que la generación de elementos basada en la simulación real de fenómenos

es, en general, demasiado costosa y complicada como para satisfacer también el resto de requisitos.

3.2.2 Requisitos de los generadores de mundos virtuales volumétrico más utilizados en la industria

Como ya hemos visto, existen diferentes acercamientos a la generación procedural de mundos virtuales en la industria (sección 2.3), debido a ello, para realizar este análisis vamos a tener en cuenta exclusivamente aquellos generadores de mundos virtuales volumétricos más conocidos (*Minecraft* [Moj11], *Terraria* [Re-11] y *Starbound* [Chu13].

3.2.2.1 Innovación

El nivel de innovación de los generadores más utilizados en la industria depende mucho de su objetivo. Por un lado, tenemos los generadores de mundos virtuales de *Terraria* [Re-11] y *Minecraft* [Moj11], en los cuales todos los mundos virtuales generados son similares y presentan los mismos elementos. Por otro lado, el generador de *Starbound* [Chu13] tiene una innovación mayor, ya que la variación entre los resultados es enorme.

Además, la innovación en los tres generadores se ve resentida, ya que la generación de mazmorras se realiza mediante el ensamblado de partes predefinidas, lo que crea estructuras repetidas y, por tanto, predecibles.

3.2.2.2 Usabilidad, control, ampliabilidad

Estos generadores están completamente focalizados en su objetivo específico y están cerrados a usuarios externos, por lo que no contemplan aspectos como la usabilidad, el control o la ampliabilidad.

3.2.2.3 Escalabilidad

Estos generadores siempre obtienen resultados a la misma escala, por lo que su escalabilidad es nula.

3.2.2.4 **Estructura**

Los generadores de *Terraria* [Re-11] y *Starbound* [Chu13] obtienen resultados con un nivel de estructura muy alto. Por su parte, el generador de *Minecraft* [Moj11], en algunas ocasiones, genera elementos fuera de lugar (árboles en agujeros imposibles, riachuelos manando de la nada o desiertos junto a zonas glaciales) que dan la impresión de cierta aleatoriedad.

Una característica común en la estructura de los resultados de todos los generadores volumétricos de la industria es que sus terrenos están siempre constituidos en capas sedimentarias de diferentes materiales.

3.2.2.5 **Interés**

Debido a las características jugables de las aplicaciones presentes en la industria todos los generadores obtienen resultados con un interés muy alto.

A pesar de ello, el interés se ve afectado a la hora de crear mazmorras en *Terraria* [Re-11] y *Starbound* [Chu13] ya que se construyen mediante estructuras predefinidas por los diseñadores, hecho por el cual se repiten muy a menudo afectando a la experiencia jugable. *Minecraft* [Moj11] también ve resentido este aspecto a la hora de generar mazmorras, ya que estas no están construidas en base a la jugabilidad.

Otro problema de los generadores de la industria es el interés de las cuevas que generan, ya que solo *Terraria* [Re-11] motiva su exploración mediante eventos como tesoros o habitaciones con diferentes recursos.

Las características comunes para todos los generadores volumétricos de la industria relacionada con este aspecto son que todos ellos generan mazmorras explorables que plantean retos jugables al jugador (encuentros con enemigos, tesoros, puzles...), y que almacenan los recursos objetivo del jugador (hierro, oro, diamante...) en pequeñas acumulaciones a diferentes profundidades difíciles de encontrar.

3.2.2.6 **Velocidad**

El generador de *Terraria* [Re-11] se ejecuta de manera *offline*, aunque no es supervisado y la generación se realiza en el sistema del usuario (ejecución híbrida). El generador de *Starbound* [Chu13] también es del mismo tipo, aunque disfraza el tiempo de generación del mundo virtual como el tiempo necesario para viajar por el espacio hasta dicho planeta.

Por otro lado, el generador de *Minecraft* [Moj11] se ejecuta completamente *online*.

3.2.2.7 **Realismo**

Ninguno de los generadores analizados intentan obtener mundos realistas, pero todos ellos intentan simular estructuras que sí son reales. Por ejemplo, todos ellos crean islas flotantes, aunque las obtienen con un aspecto realista.

3.2.2.8 **Conclusión**

Debido a las características del entorno empresarial, los generadores presentes en la industria basan su funcionamiento en la practicidad e interés de sus resultados con sus propios objetivos como meta, manteniendo el resto de características a unos niveles medios para que no interfieran en la experiencia jugable.

3.2.3 **Requisitos finales**

Finalmente, y habiendo analizado tanto las características deseables de los generadores de la literatura como las peculiaridades de los generadores de mundos virtuales presentes en la industria, obtenemos ciertas conclusiones y establecemos el nivel mínimo de nuestra métrica que debe cumplir un generador procedural de terrenos para cada uno de sus aspectos críticos. Podemos ver un resumen de estos requisitos en la tabla 3.1.

3.2.3.1 **Innovación**

El nivel de innovación, llamativamente dispar en la industria, es uno de los aspectos sobre los cuales giran todos los generadores procedurales de cualquier tipo, por lo que la innovación de los resultados debe ser lo más alta posible (nivel **2**). Debido a las características volumétricas de los terrenos objetivo de esta tesis, esta innovación debe estar presente en todos los elementos del mundo virtual, y no solo en los accidentes geográficos del terreno.

3.2.3.2 **Usabilidad**

La usabilidad de los generadores es uno de los requisitos que más atención recibe desde el ámbito académico, por lo que en establecemos este requisito en el nivel más restrictivo

(nivel **1**).

3.2.3.3 **Control**

Del mismo modo que la usabilidad, el control sobre el resultado es uno de los requisitos que más atención recibe desde el ámbito académico, por lo que lo establecemos en el nivel más restrictivo, aunque evitando que entre en conflicto con otros requisitos. Debido a que la velocidad de generación debe ser *online* o híbrida es imposible tener un método interactivo (lo que sería un nivel **3** de control), por lo que el requisito para este aspecto es de nivel **2**.

3.2.3.4 **Ampliabilidad**

Atendiendo a los requisitos impuestos por la literatura y con el objetivo de aumentar la adaptabilidad del método a diferentes objetivos y condiciones, el nivel de restricción para este requisito es el mayor posible (nivel **2**). Esto supone que la adaptación del proceso para obtener diferentes tipos de mundos virtuales debe realizarse mediante modificaciones simples de los parámetros de entrada.

3.2.3.5 **Escalabilidad**

Con el mismo objetivo que el apartado anterior, y debido a los mismos argumentos, el método de generación debe tener una escalabilidad de nivel **1**.

3.2.3.6 **Estructura**

La estructura de los resultados es la característica principal que diferencia un mundo virtual adecuadamente generado con el simple ruido aleatorio. Debido a ello, establecemos como requisito para esta característica el máximo nivel posible(nivel **2**). Además, atendiendo a los requerimientos de la industria, los terrenos deben estar estructurados en capas de materiales.

3.2.3.7 **Interés**

Debido a que la investigación de esta tesis intenta explorar un método válido tanto para la industria como para el ámbito académico, el interés de los mundos virtuales generados debe ser lo más alto posible (nivel **2**). Además, y debido a las características de

los generadores de la industria, el método de generación debe crear elementos explorables que planteen retos adicionales al jugador, como mazmorras o cuevas estructuradas teniendo en cuenta diferentes parámetros jugables establecidos por el diseñador (encuentros con enemigos, recompensas en forma de tesoro, entradas a la mazmorra...). Además, debe disponer de algún sistema para determinar qué recursos deben estar distribuidos y limitados, debido a que son importantes para el jugador

3.2.3.8 **Velocidad**

Aunque la literatura no establece ningún requisito de velocidad, la industria sí impone un requisito claro: la generación debe realizarse de manera completamente independiente para ofrecer una experiencia jugable diferente en cada partida. Por ello, nuestra investigación debe permitir la generación en el sistema del usuario (como mínimo, debe presentar una generación híbrida [nivel **1**]).

Característica crítica	**Requisito**	**Otros requisitos**
Innovación	2	
Usabilidad	1	
Control	2	
Ampliabilidad	2	
Escalabilidad	1	
Estructura	2	· Terrenos formados por capas de materiales.
Interés	2	· Estructuras jugables adicionales, como cuevas o mazmorras. · Vetas de mineral que distribuyan recursos jugables.
Velocidad	1	
Realismo	2	

Tabla 3.1: Resumen de los requisitos.

3.2.3.9 **Realismo**

El realismo absoluto solamente es posible conseguirlo mediante simulaciones formales de fenómenos. A pesar de ello, tanto el mundo industrial como el académico coinciden en que los generadores que imitan la realidad en vez de simularla obtienen resultados

suficientemente realistas. Debido la coincidencia de las dos vertientes de este campo, establecemos el requisito de realismo en un nivel **2**.

«¡Voy a continuar mi fascinante investigación sobre las maravillosas propiedades de la arena!»

CL4P-TP (*Borderlands*, Gearbox Software, 2009)

CAPÍTULO

4

Representación volumétrica del terreno

ANTES de describir el proceso de generación en sí mismo, dedicamos este capítulo a la representación de terrenos volumétricos utilizada en todos los experimentos de esta tesis. Representar un conjunto de datos volumétricos es una cuestión recurrente para diferentes áreas científicas y técnicas, como la informática gráfica, el diseño de interfaces de usuario o la visualización de datos médicos. La manera clásica para representar este tipo de datos es utilizar una matriz tridimensional de *voxels*, en la que cada elemento representa los datos volumétricos de un punto del espacio. A pesar de ello, existen otras maneras de representarlos, como la propuesta por Benes y Forsbach [BF01], que crea un sistema de capas para simular la erosión de los elementos en un terreno volumétrico, o la utilizada por Muller *et al.* [MCG03] que representa volúmenes líquidos y viscosos mediante un sistema de partículas.

El sistema de representación de datos utilizado es uno de los factores principales que condicionan el rendimiento y la eficiencia en el almacenamiento, acceso y modificación de un conjunto de datos volumétricos. Además, su viabilidad es completamente dependiente del objetivo del volumen a representar y del propósito de la aplicación. Por ejemplo, un sistema de representación que se basa en detallar el volumen al máximo puede ser perfecto para una aplicación de análisis y minería de datos, pero completa-

mente inútil si la aplicación objetivo requiere una interactividad en tiempo real con el usuario.

Lo que resta de capítulo se organiza de la siguiente manera: en la sección 4.1 se enumeran y se analizan las características principales del sistema de representación que utilizamos en esta investigación, en la sección 4.2 se realiza una descripción en profundidad del método de representación y en la sección 4.3 se presenta una comparación de nuestro método con una representación tradicional basada en *voxels*.

4.1 Características principales

Antes de introducir el sistema de representación de volúmenes utilizado en esta investigación, vamos a enumerar sus características principales. Este conjunto de características están directamente relacionadas con los requisitos definidos en el capítulo 3.

4.1.1 Facilidad para gestionar capas

El método de representación tiene una estructura que facilita el almacenamiento de terrenos basados en capas horizontales de diferentes materiales y características, simplificando la generación de terrenos basados en capas. Esto ayuda a que el método de generación obtenga un terreno estructurado y basado en la realidad.

4.1.2 No limitar las estructuras representables

El sistema de representación de volúmenes no limita el tipo de estructuras representables para no influir en la innovación y ampliabilidad del método de generación. De esta manera, la creatividad del algoritmo de generación es total y está únicamente limitada por sus características técnicas.

4.1.3 Permitir cuevas y zonas vacías

Las cuevas son un elemento importante para los generadores de terrenos volumétricos de la industria. El sistema que proponemos permite que el algoritmo de generación defina sistemas de cuevas y otro tipo de zonas vacías de material. Esta característica afecta tanto al realismo de los resultados, como al nivel de interés de los mundos virtuales de los videojuegos basados en la exploración.

4.1.4 Rapidez en la gestión de los datos

Uno de los aspectos principales para determinar la calidad de un sistema de representación es la velocidad de acceso a los datos que almacena. Como veremos más adelante, el sistema que proponemos solamente requiere unas pocas operaciones para ello, permitiendo una interacción rápida con el terreno almacenado.

4.1.5 Tamaño del terreno reducido

El otro aspecto principal de los sistemas de representación de datos volumétricos es su capacidad para reducir el tamaño de los datos. Este aspecto es crítico debido a que el objetivo de esta investigación es generar terrenos con los cuales después existirá una interacción. Por ello, nuestro sistema reduce sensiblemente la cantidad de espacio requerido en memoria frente a los sistemas de *voxels* tradicionales, como podemos ver más adelante en sección 4.3.

4.1.6 Acceso transparente a los datos

A pesar de que en el capítulo 3 solo se hace referencia a la usabilidad del método de generación de terrenos, es importante que el método utilizado para representar el terreno mantenga también un nivel alto de usabilidad a la hora de interactuar con él. El sistema que proponemos en esta investigación oculta todos los aspectos técnicos al usuario, permitiendo acceder a los datos volumétricos como si de un sistema de *voxels* tradicional se tratara.

4.1.7 Terrenos teóricamente infinitos

Nuestro método de representación no limita de ninguna manera el tamaño que un terreno pueda tener (aunque la limitación impuesta por el *hardware* sigue existiendo), lo que facilita la generación en tiempo real del terreno, ya que puede extenderse infinitamente.

4.1.8 Mezclas de materiales

El sistema de generación permite, de manera nativa, obtener terrenos cuyos materiales se fundan progresivamente, lo que aumenta el realismo final de los terrenos, otorgán-

doles un aspecto mucho más natural al evitar bordes excesivamente marcados.

4.2 Método de representación

El método de representación utilizado en esta tesis está basado en el método introducido por Benes y Forsbach [BF01], en el cual se establecen las características del terreno como una acumulación vertical de diferentes materiales formando capas. Para comprender correctamente el funcionamiento del sistema propuesto es necesario conocer previamente una serie de conceptos desde el punto de vista de nuestro sistema de representación.

4.2.1 Conceptos previos

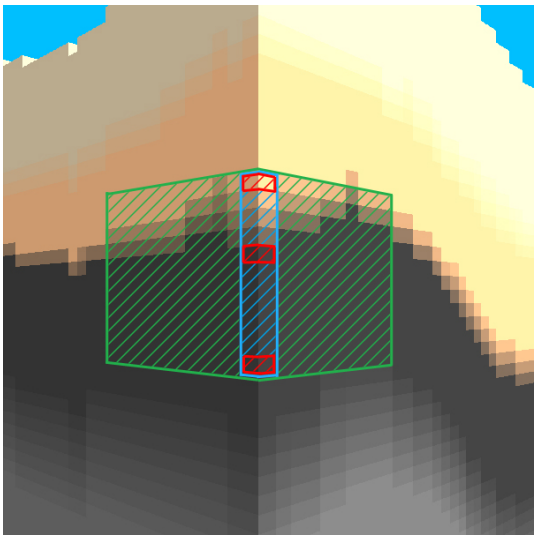


Figura 4.1: *Chunks*, columnas y límites.

4.2.1.1 Terreno

Desde la perspectiva de nuestro método de representación de volúmenes, un terreno es una colección de *chunks* (elemento que representa un pedazo tridimensional de terreno [ver sección 4.2.1.4]), además de una colección de todos los materiales (ver sección 4.2.1.2) existentes en dicho terreno junto a sus características. En la figura 4.1 podemos ver una parte de un terreno.

Un terreno puede tener un número indeterminado de *chunks* (incluso 0, aunque en este caso el terreno estaría vacío), no siendo obligatorio que todos los que posea estén juntos en el espacio. Es perfectamente posible que un terreno contenga *chunks* de dos regiones distantes en el espacio, mientras que los *chunks* del resto del terreno no existan. Esta característica facilita que el terreno pueda ser construido de manera *online* mientras varios jugadores lo recorren de manera independiente. El acceso a los *chunks* de un terreno se realiza mediante sus coordenadas (x, y, z).

Formalmente, un terreno se define con la ecuación 4.A, donde T es el terreno; m, n y u son las dimensiones de dicho terreno, cuyo rango comprende desde $-\infty$ hasta ∞, siendo su único límite las limitaciones técnicas del *hardware*; y $\mathscr{M}_{m\times n\times u}$ es la matriz tridimensional de *chunks* que conforma el terreno donde no todos sus elementos tienen la obligación de existir.

$$T \in \mathscr{M}_{m\times n\times u} \tag{4.A}$$

4.2.1.2 **Material**

Los materiales son aquellos elementos básicos que forman los datos volumétricos del terreno (*Minecraft* [Moj11], por ejemplo, utiliza tierra, piedra y arena entre otros). Los materiales se definen por sus características físicas, básicamente, un color y una textura. En la figura 4.1 podemos ver un terreno volumétrico formado por diferentes materiales.

4.2.1.3 ***Voxel***

Un *voxel* es la mínima unidad volumétrica tridimensional. Son pequeños cubos organizados en una matriz tridimensional regular que almacenan la información del volumen en dicho punto. Es el método convencional para almacenar información volumétrica, aunque en nuestro sistema de representación solamente son el medio mediante el cual se accede a dicha información por una entidad externa. Este proceso se explica en profundidad en la sección 4.2.2. En la figura 4.1 los *voxels* son todos aquellos cubos de los que está formado el terreno (podemos diferenciarlos en los cortes laterales porque cada *voxel* tiene un color diferente).

4.2.1.4 *Chunk*

Un *chunk* es un cubo regular que contiene la información volumétrica de una sección del terreno compuesta por *voxels*. Si la longitud del lado del *chunk* es de *n voxels* de longitud, este encapsula la información de una región del terreno cúbica de n^3 *voxels*. La manera en que se encapsula toda esta información dentro de un *chunk* es mediante columnas (ver el punto 4.2.1.5). Para ello, cada *chunk* contiene n^2 columnas, que representan las columnas de *voxels* dentro del *chunk*. En la figura 4.1 podemos ver un *chunk* pintado de verde con una de sus columnas pintada de azul.

Matemáticamente, un *chunk* se define mediante la ecuación 4.B, donde C_{xyz} es el *chunk* situado en las coordenadas (x, y, z); n es la longitud del lado del *chunk* en *voxels*; y $\mathscr{M}_{n\times n\times n}$ es la matriz tridimensional de *voxels* pertenecientes al *chunk*.

$$C_{xyz} \in \mathscr{M}_{n\times n\times n} \tag{4.B}$$

A pesar de que para cualquier entidad externa un *chunk* se define con la ecuación 4.B, en realidad un *chunk* se define internamente mediante la ecuación 4.C, donde C_{xyz} es el *chunk* situado en las coordenadas (x, y, z); n es la longitud del lado del *chunk* en *voxels*; y $\mathscr{M}_{n\times n}$ es la matriz bidimensional de columnas pertenecientes al *chunk*.

$$C_{xyz} \in \mathscr{M}_{n\times n} \tag{4.C}$$

4.2.1.5 **Columna**

Como ya hemos dicho previamente, un *chunk* está formado por n^2 columnas, siendo n la longitud del lado del *chunk* en *voxels*. Estas columnas, por tanto, contienen la información de *n voxels*, representando la acumulación de material de manera vertical en cada punto horizontal del *chunk*.

A pesar de ello, ya hemos dicho que internamente no se utilizan *voxels* para almacenar los datos volumétricos. En nuestro sistema esta acumulación de material se define mediante un conjunto ordenado de límites (ver sección 4.2.1.6).

En la figura 4.1 podemos ver una columna pintada de azul con sus límites pintados de rojo. Esta columna es una de las columnas del *chunk* pintado de verde de esa misma figura.

Matemáticamente, una columna se define mediante la ecuación 4.D, donde L_{xz} es

la columna situada en las coordenadas (x, z) del plano horizontal; n es la longitud de la columna en *voxels*, que es igual que la longitud del lado del *chunk* que la contiene; y $\mathscr{M}_n$ es la matriz de *voxels* que conforman la columna.

$$L_{xz} \in \mathscr{M}_n \tag{4.D}$$

A pesar de que para cualquier entidad externa una columna se define mediante la ecuación 4.D, en realidad es el conjunto ordenado de límites definido mediante la ecuación 4.E, donde L_{xz} es la columna situada en las coordenadas (x, z) del plano horizontal; l son los límites de su interior; $Límites_{xz}$ es el conjunto de límites con las mismas coordenadas (x, z) en el plano horizontal que la columna L_{xz}; h_l es la altura del límite l; h_c es la altura del *chunk* en el cual se encuentra; y n es la longitud del lado del *chunk* en *voxels*.

$$L_{xz} = \{l | l \in Límites_{xz}, h_l < (h_c + n), h_l \geq h_c\} \tag{4.E}$$

El orden del conjunto de límites de las columnas se define mediante la ecuación 4.F, donde a y b son límites de la columna L_{xz}, y h_a y h_b son las alturas de los límites a y b respectivamente.

$$\forall a, b \in L_{xz} : a < b \leftrightarrow (h_a \leq h_b) \& (a \neq b) \tag{4.F}$$

A su vez, el conjunto $Límites_{xz}$ se define mediante la ecuación 4.G, donde l son los límites de su interior; $Límites$ es el conjunto de todos los límites del terreno; x_l y z_l son las coordenadas en el plano horizontal del límite; y x y z son las coordenadas en el plano horizontal de la columna.

$$Límites_{xz} = \{l | l \in Límites, x_l = x, z_l = z\} \tag{4.G}$$

4.2.1.6 **Límite**

Desde el punto de vista de nuestro método de representación de volúmenes, un límite es el tipo de material que hay en una altura dada de una columna. Este concepto recibe la denominación de límite ya que debido al sistema de acceso a la información (ver sección 4.2.2), delimitan los límites entre los diferentes tipos de material de cada columna.

La única restricción de este sistema es que debe existir siempre un límite tanto en el punto inferior como el superior de todas las columnas. En la figura 4.1 vemos varios límites pintados de rojo que pertenecen a la columna pintada de azul. Además, podemos apreciar que los límites coinciden con los cambios de material vertical de la columna (en este caso cambia el color), así como con el punto superior e inferior de la misma (el efecto de degradado de la sección entre los dos límites superiores se produce debido al método por el cual se accede a la información volumétrica, explicado en la sección 4.2.2).

4.2.2 Acceso a los datos volumétricos

Todo este sistema de límites, columnas y *chunks* es transparente para el usuario al acceder a los datos volumétricos. Cuando una entidad externa utiliza el sistema que proponemos, accede a los datos de la misma manera que accedería a los *voxels* en un sistema tradicional. Para ello, con solo indicar las coordenadas (x, y, z) del *voxel* a consultar, el sistema accederá al *chunk* y a la columna correspondiente. En este se pueden dar dos situaciones diferentes.

Por un lado, es posible que la columna contenga un límite exactamente a la misma altura que el *voxel* que se desea acceder. En ese caso, se construye dinámicamente un *voxel* con los datos de dicho límite, sin necesitar ningún tipo de cálculo.

Figura 4.2: Columna de ejemplo.

Por otro lado, si la columna no contiene un límite a la misma altura que el *voxel* al que se desea acceder, el sistema realiza una interpolación entre los límites inmedia-

tamente superior e inferior, calculando el valor dinámicamente. Esta interpolación se realiza para todas las propiedades volumétricas (explicadas en el punto 4.2.3) de manera lineal. En la figura 4.1 podemos apreciar como se obtienen los *voxels* entre límites (la figura 4.2 muestra como está configurada la columna renderizada en esta figura). De esta manera, cuando se accede a una posición entre dos límites con las mismas características, el *voxel* obtenido tiene también las mismas características que los límites (zona inferior de la columna de ejemplo), mientras que si los límites son diferentes, el *voxel* obtenido combina los atributos de ambos límites teniendo en cuenta la cercanía a la que se encuentra cada uno, obteniendo así un degradado progresivo de los atributos (zona superior de la columna de ejemplo).

4.2.3 **Datos volumétricos**

Los límites almacenan la información volumétrica en puntos clave del terreno, generando dinámicamente la información volumétrica al completo. La naturaleza de esta información puede variar dependiendo del objetivo del terreno, ya que dependiendo de para qué se vaya a utilizar puede ser necesario almacenar conceptos tan dispares como el color, la viscosidad del material o sus características lumínicas. A pesar de ello, en esta investigación se han seleccionado una serie de características clave que ayudan a representar el terreno generado posteriormente. Estas características son las siguientes:

4.2.3.1 **Combinación de materiales**

Cada punto del terreno está formado por una combinación de materiales y a su vez, estas combinaciones están formadas por una colección de parejas de cada material presente y su concentración en dicho punto del volumen. La única restricción para estas combinaciones es que la suma de las concentraciones de todos los materiales presentes en cada punto debe resultar igual a 1.

Además, cada material contiene una serie de atributos físicos que definen las propiedades de dicho material. Combinando los atributos de los materiales de cada punto volumétrico se obtienen los atributos globales de dicho punto. Esta combinación se realiza mediante la ecuación 4.H, donde A_i es el valor del atributo global i; N es el número de materiales en el punto; A_{ij} es el valor del atributo i del material j; y C_j es la concentración del material j.

$$A_i = \sum_{j=0}^{N} (A_{ij} \cdot C_j) \tag{4.H}$$

Estos atributos son los siguientes:

- **Color:** Define el color de cada punto volumétrico del terreno.
- **Textura:** Define la textura de cada punto volumétrico del terreno. La combinación de texturas se realiza mediante *blending* de texturas.

4.2.3.2 **Flujo de material interno**

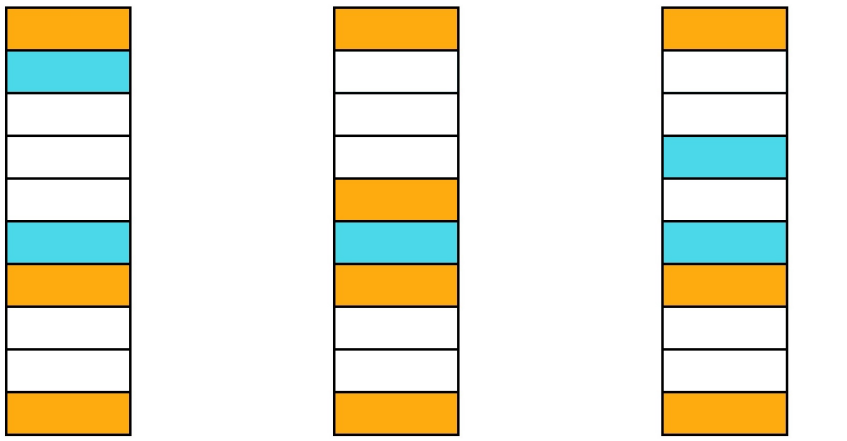
El flujo determina hacia donde está estirado el material en dicho punto del terreno.

En nuestro sistema, se ha utilizado esta información para transformar la textura del volumen en dicho punto, estirándola en la dirección que indica el vector que determina el flujo.

4.2.4 **Zonas vacías**

La representación de zonas de aire o vacías es una característica necesaria si se quiere representar terrenos más complejos que simples ortoedros formados por mezclas de materiales. Las zonas vacías permiten visualizar la superficie superior del terreno, así como estructuras subterráneas como cuevas o canales. Debido a las particularidades de nuestro sistema abordamos este tipo de zonas utilizando un material vacío que representa al aire. Las restricciones de este material son las siguientes:

- Un límite que contenga un material de aire no puede contener más materiales adicionales. Por consiguiente, la concentración del material vacío tiene que ser siempre 1.
- No se pueden interpolar *voxels* a partir de un límite con material vacío y otro con materiales sólidos, debido a que esto generaría *voxels* que mezclan materiales sólidos normales con el material vacío, algo imposible en nuestro sistema. Esto implica que después de un límite con material vacío solamente puede haber otro límite con material vacío dentro de una columna, a no ser que estén en alturas consecutivas.



(a) Columna correcta (b) Columna correcta (c) Columna incorrecta

Figura 4.3: Columnas conteniendo zonas de vacío.

En la figura 4.3 podemos ver tres columnas representadas como una serie de límites apilados. Los límites de color azul son límites con el material vacío; los límites de color naranja contienen cualquier otro tipo de composición de materiales; por último, los límites de color blanco no existen, y en caso de acceder a dicha posición el *voxel* se interpolaría entre el límite superior e inferior. Las subfiguras 4.3(a) y 4.3(b) representan columnas correctas, ya que es imposible interpolar un *voxel* entre un límite vacío y uno sólido. Por el contrario, la subfigura 4.3(c) representa una columna incorrecta, ya que entre el límite más alto (sólido) y el segundo límite más alto (vacío) existen *voxels* que deben interpolarse. Para solucionar este problema hay que decidir si esta zona debe vaciarse (en cuyo caso será necesario mover el límite con material vacío a la altura inmediatamente inferior a la que se encuentra el material más alto, quedando la columna igual que la que se muestra en 4.3(a)), o rellenarse con material (en cuyo caso debería crearse un límite nuevo con material sólido en la altura inmediatamente superior a la que se encuentra el límite con material vacío más alto).

4.2.5 *Render*

Aunque este procedimiento queda fuera del alcance de esta tesis, el sistema propuesto tiene ciertas peculiaridades que permiten facilitar el proceso de *render*. Además, las características volumétricas de cada punto afectan al renderizado directamente.

4.2.5.1 **Peculiaridades en el renderizado**

El renderizado es uno de los aspectos principales de todos los campos científicos y técnicos relacionados con la informática gráfica. Debido a ello, y a pesar de que queda fuera del alcance de esta tesis, vamos a analizar levemente las implicaciones que tiene en el proceso de *render* el sistema de representación que utilizamos en nuestra investigación.

La principal ventaja de nuestro sistema de representación de terrenos volumétricos a la hora de renderizarlos es que no hace falta acceder a todos sus *voxels* para determinar las características internas y externas del volumen. Únicamente es necesario acceder a los límites, y a partir de ellos deducir las características del resto de material. Este hecho tiene dos consecuencias principales:

- **Descartar los *chunks* innecesarios:** Es común que un terreno volumétrico esté formado tanto por *chunks* internos del terreno como por *chunks* que solo contienen aire. Normalmente, estos *chunks* de aire no se renderizan (aunque hay excepciones que ser, por ejemplo, mundos virtuales volumétricos con zonas de gases semitransparentes), y es conveniente discriminarlos previamente para liberar de dicho proceso al procedimiento de renderizado. En un sistema de representación de volúmenes tradicional sería necesario acceder a los *voxels* de cada *chunk* de manera secuencial hasta que se encuentre un *voxel* que no sea de aire o hasta que se recorra el *chunk* al completo. Por el contrario, con nuestro sistema de representación, solamente es necesario acceder a los límites, ya que si no existe ningún límite de aire en el *chunk* significa que no existen *voxels* de aire en él. La ganancia de rendimiento en el proceso es dependiente del número de límites del cada *chunk*, aunque en el peor de los casos el número de límites a consultar es exactamente el mismo que el número de *voxels* a consultar en un sistema tradicional.

- **Determinar los *voxels* externos mediante los límites:** Otra característica común a la mayoría de sistemas de renderizado de volúmenes es que no es necesario renderizar aquellos *voxels* que se encuentran completamente rodeados por otros *voxels*, y a que no son visibles (al igual que en el punto anterior existen excepciones a esta regla, como el renderizado de volúmenes semitransparentes representando, por ejemplo, cristal o hielo). En los sistemas tradicionales, para determinar qué *voxels* son externos y por consiguiente deben renderizarse, es necesario acceder a los seis *voxels* que rodean cada *voxel*. Por el contrario, en nuestro sistema este proceso puede realizarse por bloques, ya que cada pareja de límites determinan un rango de *voxels* dentro de cada columna. Al igual que en el punto anterior, la

ganancia de rendimiento con nuestro sistema es dependiente del número de límites y de la distancia entre ellos, y, en el peor de los casos, se obtendría el mismo rendimiento que en un sistema de representación de volúmenes tradicional.

Además de determinar qué *voxels* son externos y, por tanto, deben renderizarse, es posible aumentar el rendimiento del sistema discriminando aquellas caras de los *voxels* que están tapadas por otros. Este proceso se realiza de la misma manera que el proceso anterior, pudiendo incluso llevarse a cabo a la vez.

4.2.5.2 **Influencia de los datos del volumen**

Los datos volumétricos utilizados en nuestro sistema influyen directamente en el renderizado del terreno. Esta influencia no es innata al método de representación de volúmenes que proponemos, ya que es directamente portable a un sistema de representación de volúmenes tradicional basado exclusivamente en *voxels*.

- **Color:** El color de cada *voxel* viene dado por los colores de los materiales combinados en el punto en cuestión, además de por la concentración de estos materiales. Como ya hemos visto, este cálculo se realiza mediante la ecuación 4.H, que se ejecuta directamente en la GPU (*Graphics Processing Unit* o Unidad de Procesamiento Gráfico) para ahorrar ciclos de la unidad de proceso central.
- **Textura:** La textura de cada *voxel*, al igual que el color, viene dada por las texturas de los materiales combinados en el punto en cuestión, además de por la concentración de estos materiales. Este cálculo también se realiza mediante la ecuación 4.H en la GPU. La textura se pintará siempre encima del color del *voxel*, permitiendo ver el color en función de la transparencia (el canal alfa) en cada píxel.
- **Flujo:** Es posible utilizar la información del flujo de material de múltiples formas en el proceso de renderizado, como la deformación de la geometría del *voxel* o la rotación de su textura. En nuestro sistema estos datos se utilizan para deformar la textura para que se estire en la dirección y con la intensidad que marca el flujo.

Como hemos dicho, el proceso encargado de combinar tanto el color como la textura de los materiales (mediante la ecuación 4.H) se ejecuta en la GPU. A continuación se muestra el código encargado de determinar las características de cada punto mediante los materiales y de combinar la textura con el color. Está programado en lenguaje HLSL e incluye los parámetros del *shader*, los *samplers* y las funciones involucradas en el proceso, además de la sección del *pixel shader* que se encarga de llevarlo a cabo.

```
//#########################################################
//#                  Shader parameters                    #
//#########################################################

// Number of materials
int materialNumber;

// Textures
Texture2D materialTexture[10];

// Colors
float4 materialColor[10];

// Concentrations
float materialConcentration[10];



//#########################################################
//#                       Samplers                        #
//#########################################################

SamplerState simpleSampler
{
    Filter = MIN_MAG_MIP_LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};



//#########################################################
//#                      Functions                        #
//#########################################################

// Function: GetColor
//   texture: Material texture
//   color: Material color
//   concentration: Material concentration
//   texCoord: Texture coordinates
// Return: The color for this material and
// concentration
float4 GetColor( Texture2D texture, float4 color, float concentration,
    float2 texCoord)
{
    // Get the color
    float4 tcolor =  texture.Sample(simpleSampler, texCoord);
    tcolor = ( 1 - tcolor.a) * color + tcolor.a * tcolor;

    // Apply the concentration
    return tcolor * concentration;
}
```

```
//#########################################################
//#                      Pixel shader                     #
//#########################################################

float4 PS(VertexShaderOutput input) : SV_TARGET
{

    ...

    // Get the combined color
    float4 color = float4(0, 0, 0, 1);
    for(int i = 0; i < materialNumber; i++)
        color += GetColor(materialTexture[i], materialColor[i],
            materialConcentration[i], input.TexCoord);

    ...

}
```

4.2.6 Almacenamiento de la información

El último aspecto a analizar del sistema de representación de volúmenes utilizado es el almacenamiento de toda la información anteriormente descrita en disco. Para ello se utiliza una estructura de ficheros de texto formada por los siguientes ficheros:

- **Fichero de terreno:** Fichero que hace referencia a todo el terreno. Contiene los siguientes campos:
  - Nombre del terreno
  - Tamaño del *chunk* expresado como la longitud de su lado en *voxels*.
  - Ruta al directorio que contiene los ficheros de *chunk*. Ruta relativa al fichero del terreno.
  - Lista con los *chunks* de los que dispone el terreno y se encuentran almacenados en disco.
- **Ficheros de *chunk*:** Los *chunks* se almacenan cada uno en su propio archivo y tienen dos áreas diferentes. La primera de ellas solamente contiene las coordena-

das (x, y, z) del *chunk*. La segunda contiene los datos de las columnas. Por cada columna se almacenan los siguientes datos:

- Coordenadas (x, z) de la columna relativas al chunk.
- Colección de límites de la columna. Los campos que definen estos límites son los siguientes:
  - Altura: Coordenada y del límite relativa al *chunk*.
  - Dirección del flujo: Componentes (x, y, z) del vector normalizado que define la dirección del flujo. Es necesario combinarlo con el módulo que indica la intensidad del flujo para obtener el vector de flujo final.
  - Intensidad del flujo: Módulo del vector de flujo. Es necesario combinarlo con el vector normalizado que indica la dirección del flujo para obtener el vector de flujo final.
  - Composición de materiales: Lista de los materiales presentes en el límite junto a su valor de concentración.

4.3 Resultados

Para medir la eficacia de nuestro sistema de representación de terrenos volumétricos vamos a comparar el tamaño en disco de los terrenos representados con nuestro sistema con terrenos representados con un sistema tradicional en el cual los *voxels* se almacenan en bruto. Debido a la cantidad variable de información que se puede guardar en un *voxel* (que depende del objetivo y las necesidades del terreno), vamos a cuantificar dicho tamaño en número de elementos volumétricos almacenados, es decir, límites en nuestro caso y *voxels* en el caso del sistema tradicional.

Para realizar esta medición almacenamos con los dos sistemas, terrenos de diferentes tamaños ($40 \times 40 \times 40$, $80 \times 80 \times 80$ y $160 \times 160 \times 160$ *voxels*), utilizando en nuestro sistema diferentes configuraciones para el tamaño de los *chunks* ($n = 5$, $n = 10$ y $n = 20$). Además, el valor obtenido por nuestro sistema varía dependiendo de la complejidad vertical del terreno, por lo que hemos realizado la medición en los dos extremos: un terreno con una variación vertical lineal y homogénea y un terreno completamente variable similar al ruido aleatorio.

La tabla 4.1 muestra los resultados de esta medición. Las filas etiquetadas como «Máximo» representan los valores obtenidos utilizando un terreno que requiera el máximo número de elementos. Por otro lado, las filas etiquetadas como «Mínimo» utilizan un terreno que requiera el mínimo de elementos volumétricos para su representación.

Tamaño del terreno	Nº de elementos	Nuestro método			Método tradicional
		$n = 5$	$n = 10$	$n = 20$	
40^3	**Máximo**	64.000	64.000	64.000	64.000
	Mínimo	25.600	12.800	6.400	64.000
80^3	**Máximo**	512.000	512.000	512.000	512.000
	Mínimo	204.800	102.400	51.200	512.000
160^3	**Máximo**	4.096.000	4.096.000	4.096.000	4.096.000
	Mínimo	1.638.400	819.200	409.600	4.096.000

Tabla 4.1: Comparación de nuestro sistema de representación con un sistema tradicional.

Podemos observar que los resultados obtenidos con nuestro método son, en el peor de los casos (que solamente se da en caso de que los datos volumétricos fuesen ruido aleatorio), exactamente iguales que los de un sistema tradicional, mientras que en el resto de situaciones la mejora es significativa. Además, la tabla muestra que la mejora es mayor utilizando un tamaño mayor para nuestros *chunks*.

«Orden, entropía... un ciclo sin fin.»

Heimerdinger (*League of Legends*, RIOT, 2009)

CAPÍTULO

5

Generación del terreno

TAL y como hemos visto en capítulos anteriores, un mundo virtual es un conjunto de diversos elementos. En este capítulo profundizamos en el proceso de generación del terreno sobre el cual construir más adelante la capa jugable. Para ello, atendemos a las características deseables definidas en el capítulo 3).

Este capítulo se estructura de la siguiente manera: en la sección 5.1 exponemos los características principales de nuestro generador; en la sección 5.4 presentamos el método y explicamos en profundidad todos sus pasos; y presentamos los resultados obtenidos por nuestro método en la sección 5.5, en la que medimos diferentes aspectos, además de exponer una muestra visual de los resultados.

5.1 Características principales

Las características principales del método de generación, al igual que las del método de representación de volúmenes, están directamente relacionadas en los requisitos de nuestra investigación, definidos en el capítulo 3. Las características deseables para el método de generación son las siguientes:

5.1.1 El resultado no es predecible

Todo proceso de generación está basado en un sistema probabilístico, por lo que es imposible predecir el resultado. Esto mejora la innovación del sistema en general, ya que se obtienen elementos inesperados con propiedades inesperadas, aunque influidos por las afinidades entre materiales, definidas por el diseñador.

5.1.2 Terrenos basados en capas de material

Los resultados obtenidos con el método de generación que proponemos están basados en la realidad, por lo que se forman mediante capas de diferentes materiales acumuladas y e radas una sobre otra.

5.1.3 Vetas de mineral

El método de generación construye vetas de diferentes recursos distribuyéndolas por todo el terreno de una manera estructurada entre las capas de material. De esta manera, el diseñador puede definir recursos importantes para el jugador, incentivando la exploración, y, por tanto, aumentando su interés.

5.1.4 Similitud entre capas consecutivas

El proceso de generación tiene en cuenta el grado de similitud que dos capas consecutivas deben tener, modificándolo en base a los materiales de cada capa. De esta manera la estructura y la cohesión del terreno aumenta, ya que se presentan las capas como un todo y no como elementos independientes. Además, la naturalidad del terreno aumenta, a la vez que mejora su realismo.

5.1.5 Afinidades e incompatibilidades entre diferentes materiales

Los materiales se relacionan entre ellos mediante un índice de compatibilidad, así, es más usual ver cerca (o incluso mezclados) aquellos materiales cuya afinidad sea alta, lo que aporta un mayor realismo al resultado y una estructura de los materiales más coherente.

5.1.6 Ejecución en tiempo real

Como veremos más adelante, el algoritmo de generación es suficientemente rápido para ejecutarse en tiempo real.

5.1.7 Extensión de un terreno previamente generado

El sistema de generación ha sido diseñado para que sea capaz de extender un terreno ya creado, lo que facilita la generación de un mundo virtual en tiempo real mientras el usuario lo explora.

5.1.8 Parámetros de entrada relacionados exclusivamente con el terreno

Todos los parámetros de entrada del método están relacionados con el terreno en sí, lo que otorga transparencia al método de generación utilizado.

5.1.9 Proceso no interactivo

El proceso de generación es completamente automático, ya que se evitan los pasos que requieren interacción con el diseñador. Aunque este hecho limita en parte el control sobre el resultado obtenido, permite que los terrenos se generen en tiempo real.

5.1.10 Información sobre el resultado exclusivamente en los parámetros de entrada

Toda la información sobre el resultado se encuentra recogida en los parámetros de entrada, es decir, el método no mantiene en su interior ningún tipo de conocimiento sobre el terreno a generar. Este hecho mejora la ampliabilidad del sistema, ya que permite variar entre diferentes tipos de terrenos modificando los datos de entrada sin necesidad de modificar el método.

5.1.11 Escala arbitraria

La escala del terreno a generar se define mediante un solo parámetro. El método es capaz de producir terrenos a cualquier escala sin ningún tipo de limitación técnica.

5.1.12 **Mezcla progresiva de materiales**

Las fronteras entre las diferentes capas y vetas subterráneas no se dividen mediante cambios bruscos de material. Estas divisiones están compuestas por cambios paulatinos entre los materiales de las diferentes capas, que otorga una mayor naturalidad y realismo al resultado.

5.2 **Materiales y métodos**

El método de generación de terrenos volumétricos propuesto en este capítulo basa su funcionamiento en dos técnicas: generadores de ruido Perlin y construcción de fractales.

5.2.1 **Ruido Perlin**

Como ya hemos visto en la sección 2.1.1, un generador de ruido Perlin [Per85] es un caso particular de generador de valores pseudoaleatorios. Esta técnica permite obtener valores que producen sensación de continuidad en el espacio (con un número de dimensiones determinado por el usuario del generador).

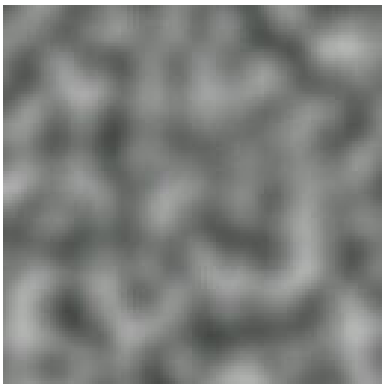


Figura 5.1: Ruido Perlin en 2 dimensiones.

La figura 5.1 muestra un ejemplo de ruido Perlin en dos dimensiones representado como una textura en escala de grises, siendo negro el más bajo y blanco el más alto de los valores obtenidos por el generador (normalmente estos valores son 0 y 1).

5.2.2 Construcción de fractales

Los fractales, como hemos visto en la sección 2.1.4.3, son construcciones formadas por versiones de sí mismas cada vez más pequeñas de manera recursiva. En nuestro caso, para la construcción de fractales utilizamos una técnica basada en el método de desplazamiento de punto medio [Mil86], algoritmo común en la generación de terrenos mediante la técnica de subdivisión de rejilla (explicada en la sección 2.1.4.2).

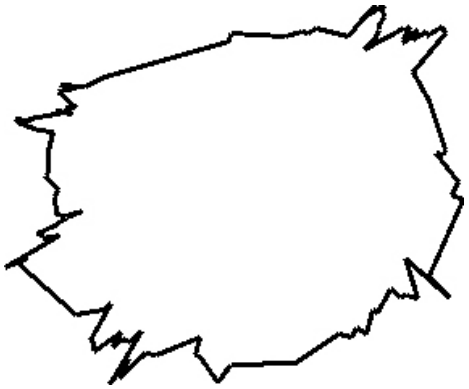


Figura 5.2: Figura fractal obtenida por nuestro método.

La particularidad de nuestro caso es que no realizamos la subdivisión en una rejilla, sino sobre los lados de una forma geométrica plana (un cuadrado antes de la primera división fractal). La figura 5.2 muestra un ejemplo de una figura obtenida con este método.

5.3 Datos de entrada

La entrada de datos de nuestro método se realiza mediante dos conjuntos diferentes de datos. Por un lado, se definen los materiales que van a aparecer en el terreno, mientras que por el otro, se determinan aspectos globales del mundo virtual.

5.3.1 Datos de los materiales

Este conjunto de datos de entrada define los materiales mediante los cuales se van a construir las diferentes capas del terreno. Para facilitar su lectura y escritura se provee a modo de fichero de texto dividido en tres secciones.

Descriptores de los materiales

En la primera sección se definen las características individuales de cada material y se determinan los siguientes atributos para cada uno de ellos:

- **Identificador:** Identificador único del material por el cual se le va a referenciar posteriormente.
- **Nombre:** Nombre del material.
- **Color:** Color del material.
- **Parámetros especiales:** Parámetros que indican ciertas características del material, por ejemplo, si permite mezclarse con otros materiales o si pertenece al estrato más bajo del terreno. Cada material puede tener uno o más parámetros especiales. Estos parámetros son los siguientes:
  - *Void:* Indica que el material no se va a renderizar y que representa las zonas huecas del terreno. Así, los materiales con este parámetro se utilizan para construir las cuevas y el aire del mundo virtual. Este parámetro, que es incompatible con cualquier otro, lleva implícito el parámetro *alone*. Además, todo conjunto de datos de entrada válido debe tener un material (y solo uno) con el parámetro *void* definido.
  - *Base:* Este parámetro indica que es un material del que va a estar compuesta la primera capa del terreno. Representa la capa conocida como roca madre.
  - *Alone:* Este parámetro señala al material que lo posee como imposible de combinar con otros materiales. Una capa que posea este material estará formada única y exclusivamente por este material.
  - *Vein:* Indica que es un material valioso para el jugador y, como tal, no debe incluirse de forma masiva en el terreno, por lo que debe limitar su aparición a vetas de mineral. Este parámetro es incompatible con el parámetro *base*.
  - *None:* Este parámetro indica que el material es un material normal. Es incompatible con cualquier otro parámetro especial.

  Podemos ver la compatibilidad entre estos parámetros en la tabla 5.1.
- **Texturas:** Textura para renderizar los *voxels* formados por este material.

	Void	*Base*	*Alone*	*Vein*	*None*
Void					
Base			✓		
Alone		✓		✓	
Vein			✓		
None					

Tabla 5.1: Compatibilidad entre los parámetros especiales.

Modificadores de los materiales

En la segunda sección se definen aquellos aspectos de los materiales que afectan a las capas de las cuales forman parte. Para cada material se definen los siguientes atributos:

- **Área:** Especifica el área relativa de las vetas de mineral formadas por este material. Está formado el área relativa mínima y el área relativa máxima.
- **Profundidad:** Profundidad relativa del terreno a la que aparecen capas formadas por el material. Está formado por la profundidad relativa mínima y la profundidad relativa máxima.
- **Grosor:** Grosor relativo de las capas formadas por el material. Está formado por el grosor relativo mínimo y el grosor relativo máximo.
- **Rugosidad:** Rugosidad relativa de las capas formadas por el material. Está formado por la rugosidad relativa mínima y la rugosidad relativa máxima.
- **Similitud:** Similitud relativa de la forma de las capas formadas por el material con la forma de la última capa del terreno. Es decir, indica cuánto se parece una capa a la superficie sobre la que se ha creado.
- **Facilidad para mezclarse:** Facilidad con la que el material se mezcla con otros material diferentes para formar una nueva capa del terreno.

Como vemos, todos los valores de esta sección son relativos (normalizados entre 0 y 1). Posteriormente, estos valores se multiplican con los parámetros globales definidos por el diseñador. Esto permite que la escalabilidad y adaptabilidad del método aumenten, gracias a que con solo cambiar los parámetros globales se modifican todos los valores de los materiales, lo que permite obtener mundos virtuales de múltiples escalas y características, con cambios leves en los datos de entrada.

Afinidad entre materiales

En la tercera y última sección se establecen las interrelaciones entre los diferentes materiales definidos en las secciones anteriores. Para cada pareja de materiales se definen los siguientes aspectos:

- **Afinidad en la misma capa:** Índice de afinidad de los materiales para mezclarse en la misma capa.
- **Afinidad en capas consecutivas:** Índice de afinidad de los materiales para mezclarse en capas consecutivas.

Estos dos índices de afinidad se encuentran normalizados entre 0 y 1, aunque existe una excepción. Para indicar que dos materiales son completamente incompatibles, esta propiedad debe contener el valor -1. La diferencia entre contener 0 y contener -1, es que con 0 no existe, a priori, ningún motivo para que estos materiales se unan, aunque es posible que un tercer material sea afín a los dos y terminen haciéndolo. Por otro lado, si entre los dos materiales existe una afinidad de -1 significa que es imposible que estos materiales se unan, aunque existan otras razones para ello.

5.3.2 Parámetros globales

El segundo y último conjunto de datos de entrada está comprendido por una serie de parámetros globales para todo el terreno. Estos parámetros definen aspectos individuales del terreno y convierten los valores relativos de los materiales en absolutos. Estos parámetros son los siguientes:

- **Profundidad del terreno:** Determina la distancia entre la primera y la última capa del terreno.
- **Grosor de capas:** Modificador global que convierte en absoluto el grosor de todas las capas que componen el terreno.
- **Grosor de vetas:** Modificador global que convierte en absoluto el grosor de todas las vetas del terreno.
- **Área de vetas:** Modificador global que convierte en absoluto el tamaño relativo del cuadrado sobre el cual se construyen las vetas del terreno.

- **Densidad de vetas:** Parámetro que determina la densidad de las vetas en el terreno.
- **Rugosidad:** Modificador global que convierte en absoluto la rugosidad de todas las capas y vetas que componen el terreno. En el caso de capas se refiere a la rugosidad de su superficie, mientras que en el de las vetas se refiere a la rugosidad del borde en el plano horizontal de las vetas.
- **Similitud de capas:** Modificador global que convierte en absoluto el grado de similitud de todas las capas con la automáticamente anterior.
- **Escala:** Parámetro que determina la escala a la cual se va a construir el terreno.

5.4 Proceso de generación

El proceso de generación del terreno se divide en tres pasos principales. Primero, el método obtiene los datos de entrada definidos por el diseñador y los procesa adecuándolos e indexándolos, para facilitar y agilizar pasos posteriores. A continuación, se realiza la verdadera generación del terreno, aunque a un nivel de abstracción mayor que el sistema de representación definido previamente. Para finalizar, el método convierte el resultado del paso anterior al sistema de representación de datos volumétricos utilizado en nuestra investigación.

5.4.1 Procesamiento de los datos de entrada

Debido a las características del método, los datos de entrada deben ser adecuadamente procesados e indexados. Por un lado, los materiales se indexan a partir de sus parámetros especiales, con el fin de acelerar el paso de selección de materiales para la generación de capas y vetas de mineral. Por otro lado, se crean tablas que relacionan los materiales entre sí, pudiendo determinar las afinidades e incompatibilidades de cada material de una manera sencilla.

5.4.2 Generación del terreno

El algoritmo de generación de terrenos volumétricos propuesto en esta tesis está basado en la acumulación vertical de capas y vetas de diferentes materiales. Para ello, va generando capas y vetas de manera secuencial hasta alcanzar la profundidad especificada por el diseñador en los datos de entrada. La probabilidad para generar una veta

de mineral en vez de una capa viene dada por la densidad de vetas de mineral en el terreno, que se provee como dato de entrada para el algoritmo.

5.4.2.1 **Generación de capas**

Las capas de material de los terrenos resultantes de nuestro método están basadas en superficies creadas mediante ruido Perlin, al cual se le aplican diferentes modificadores determinados por los materiales que forman la nueva capa y la superficie del terreno sobre la cual se construye.

Este proceso se compone de los siguientes pasos:

Paso 1: Selección de los materiales

El primer paso del proceso de creación de capas es la selección de los materiales y su concentración. El proceso realizado en este paso difiere dependiendo de si la capa a construir es la primera capa del terreno o no lo es.

Si la capa a construir es la primera capa del terreno este paso se convierte en una etapa trivial, ya que se seleccionan aquellos materiales con el parámetro *base* definido. Además, la concentración de todos ellos es exactamente la misma, dada por la ecuación 5.A, donde C_i es la concentración del material i, y N es el número de materiales de la primera capa.

$$C_i = \frac{1}{N} \tag{5.A}$$

Por otro lado, si la capa a construir no es la primera capa del terreno, el proceso de selección de materiales se lleva a cabo mediante un método probabilístico, donde se tienen en cuenta factores como la afinidad de los materiales de la última capa generada o la profundidad a la que se va a crear la nueva capa. Los pasos de este proceso son los siguientes:

1. Obtención del listado de posibles materiales. No se tienen en cuenta los materiales definidos con los parámetros *base*, *void* o *vein*, o aquellos materiales incompatibles con algunos de los materiales presentes en la superficie del terreno generado hasta este momento.
2. Cálculo del índice de afinidad para cada material presente en la lista de materiales probables. Este cálculo se realiza mediante la ecuación 5.B, donde A_i es el

índice de afinidad para el material probable i, N es el número de materiales en la superficie del terreno generado hasta este momento, a_{ij} es la afinidad entre el material posible i y el material j de la superficie, y c_j es la concentración del material j en la superficie del terreno.

$$A_i = \sum_{j=0}^{N} (a_{ij} \cdot c_j) \tag{5.B}$$

3. Cálculo de la probabilidad de aparición de cada material probable en la nueva capa, que se calcula mediante la ecuación 5.C, donde P_i es la probabilidad de aparición en la nueva capa del material probable i, A_i es el índice de afinidad del material probable i calculado en el paso anterior, M es el número de materiales posibles, y A_j es el índice de probabilidad del material posible j.

$$P_i = \frac{A_i}{\sum_{j=0}^{M} A_j} \tag{5.C}$$

4. Obtención del material principal de la nueva capa. Para ello se obtiene un valor aleatorio entre 0 y 1, que se compara con las probabilidades anteriormente obtenidas y así, se selecciona uno de los materiales probables como material principal.

5. Obtención de materiales adicionales para la capa. Si el material anteriormente seleccionado tiene definido el parámetro *alone* no se seleccionan más materiales para esta capa, debido a su imposibilidad para mezclarse. Si no tiene dicho parámetro definido se ha establecido una probabilidad del 50 % de que exista un segundo material. Dicha probabilidad se divide entre dos de manera recursiva para los siguientes materiales, y cuando esta probabilidad no se supere el proceso se da por terminado. La selección de los materiales adicionales se realiza de la misma manera que la del material principal, aunque no se tienen en cuenta aquellos materiales con el parámetro *alone* definido. Además, se incluyen los materiales ya seleccionados a la hora de calcular la afinidad para los nuevos materiales probables.

6. Cálculo de la concentración de cada material. Si la capa tiene solamente un material, se le asignará concentración de 1. Si la capa está formada por dos o más materiales, al material principal se le asigna una concentración aleatoria entre 0,5 y 0,9, mientras que para el resto de materiales se divide aleatoriamente lo que resta hasta 1 (ya que la suma de las concentraciones de todos los materiales de cada capa debe ser siempre 1).

Una vez alcanzado este punto, el algoritmo ya tiene creada una lista de materiales y sus respectivas concentraciones.

Paso 2: Cálculo de los modificadores de la capa

Los modificadores de la capa se calculan utilizando los materiales previamente seleccionados y su concentración. En este paso se calcula el grosor relativo, la similitud relativa con la última capa, la facilidad relativa para mezclarse y la rugosidad relativa de la capa con la ecuación 5.D, donde V es el valor de la propiedad a calcular, N es el número de materiales de la capa, v_i es el valor de la propiedad que estamos calculando para el material i, y c_i es la concentración del material i.

$$V = \sum_{i=0}^{N} (v_i \cdot c_i) \tag{5.D}$$

Paso 3: Generación física de la capa

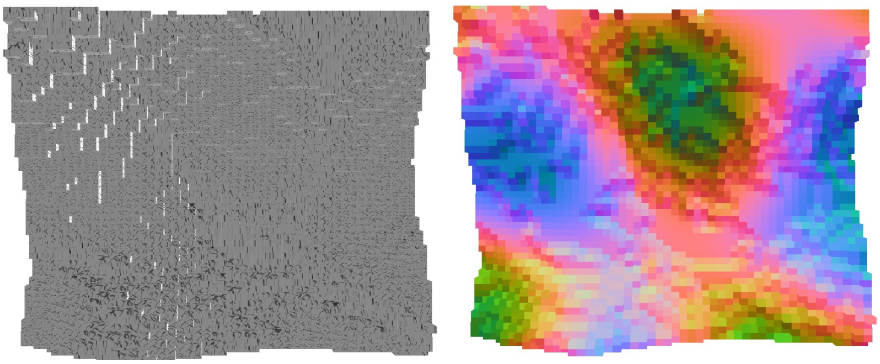
En este punto, el método genera la capa físicamente. Este paso se basa en la generación de una superficie mediante ruido Perlin [Per85], a la cual se le aplican los modificadores anteriormente calculados, así como los parámetros generales absolutos especificados en los datos de entrada. La frecuencia de dicho ruido viene determinada por la escala definida por el diseñador. A continuación, el valor obtenido por el ruido Perlin se multiplica por la rugosidad relativa de la capa, y se modifica mediante la similitud relativa con la última capa como se muestra en la ecuación 5.E, donde h es la altura modificada final, h_{-1} es la altura de la superficie del terreno generado hasta ahora en ese punto, s es la similitud relativa con la capa anterior calculada previamente, y n es el valor obtenido mediante el ruido Perlin multiplicado por la rugosidad relativa.

$$h = h_{-1} \cdot s + n \cdot (1 - s) \tag{5.E}$$

La altura obtenida en este proceso es relativa a la capa, por lo que para obtener la altura final es necesario sumarle su grosor (obtenido multiplicando el grosor relativo dado por los materiales de la capa por el parámetro general que indica el grosor de las capas) y la altura media de la capa inmediatamente anterior.

Paso 4: Generación del flujo de material de la capa

Para finalizar, el método genera el flujo de material interno de la capa. Para obtener un flujo con una apariencia realista y natural, el método lo genera tangente a la superficie de la capa en los ejes x y z, mientras que en el eje y (el plano horizontal) se rota un ángulo determinado por un generador de ruido Perlin, cuya frecuencia, al igual que en la generación de la capa en sí misma, viene dada por la escala del terreno.



(a) Vista superior de la capa.

(b) Vista superior de la capa, donde los vectores del flujo de material se representan como valores *RGB*.

Figura 5.3: Flujo de material en las capas.

En la figura 5.3 podemos ver un ejemplo de este proceso. La subfigura 5.3(a) muestra una capa vista desde arriba. Podemos apreciar como el proceso de renderizado ha deformado la textura siguiendo un patrón de ruido Perlin. Por su parte, la subfigura 5.3(b) muestra los vectores que definen el flujo de esta misma capa y los representa como valores *RGB*.

5.4.2.2 Generación de vetas

Las vetas son elementos similares a las capas, con la excepción de que en vez de extenderse hasta el infinito por el terreno, se encuentran localizadas en una zona. El proceso de generación de vetas de mineral sigue los siguientes pasos:

Paso 1: Generación del *bounding square* base

El *bounding square* es un cuadrado establecido sobre el terreno dentro del cual se construye la veta de mineral.

El *bounding square* base se establece aleatoriamente con unas dimensiones máximas establecidas por el parámetro general área de vetas, y mínimas determinadas por el valor área de vetas/2. La posición de dicho cuadrado se establece de una manera completamente aleatoria en el interior del terreno que se está generando. Este cuadrado, al aplicarle los modificadores de tamaño relativos definidos por los materiales de la veta, se transforma en el *bounding square* final.

Paso 2: Selección de los materiales

Una vez establecido el *bounding square* base se seleccionan los materiales que van a componer la veta de mineral. Este proceso es similar al proceso de selección de materiales para las capas, aunque con las siguientes diferencias:

- El procedimiento de selección de materiales se basa en las afinidades de los materiales de la superficie del terreno generado hasta ese momento, aunque en este caso solamente se tienen en cuenta las afinidades de aquellos materiales de la superficie que se encuentran justo debajo del *bounding square* base.

- El material principal para la generación de la veta se selecciona exclusivamente entre aquellos materiales que tengan el parámetro especial *vein* definido.

Paso 3: Cálculo de los modificadores de la capa

A continuación se calculan los modificadores relativos de la veta de mineral exactamente igual que en el proceso de generación de capas. Estos modificadores son el área relativa, la rugosidad y el grosor relativo.

Paso 4: Generación del *bounding square* final

El siguiente paso para la generación de vetas es el cálculo del *bounding square* final. Como hemos dicho en el primer paso de este proceso, este cuadrado se calcula partiendo del *bounding square* base. Para ello, se multiplica el *bounding square* base por el área relativa de la veta, calculada en el paso anterior. El *bounding square* final es el rectángulo sobre el terreno en el cual se construirá, finalmente, la veta de mineral.

Paso 5: Construcción de la forma de la veta

En este paso, el método construye la beta mediante un algoritmo fractal basado en la técnica de desplazamiento del punto medio, que ha sido utilizado exhaustivamente en la generación procedural de terrenos [Mil86].

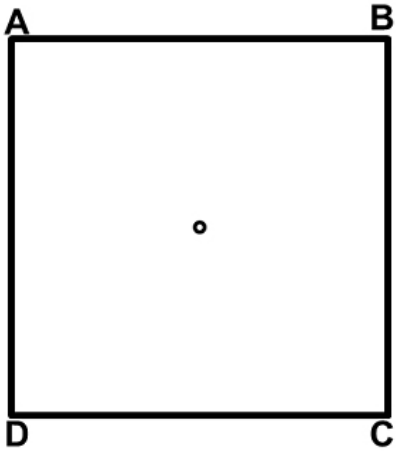


Figura 5.4: Inicio del algoritmo fractal.

El algoritmo comienza con un cuadrado $ABCD$ (figura 5.4), en el cual se desplazan sus vértices a lo largo de sus diagonales (figura 5.5).

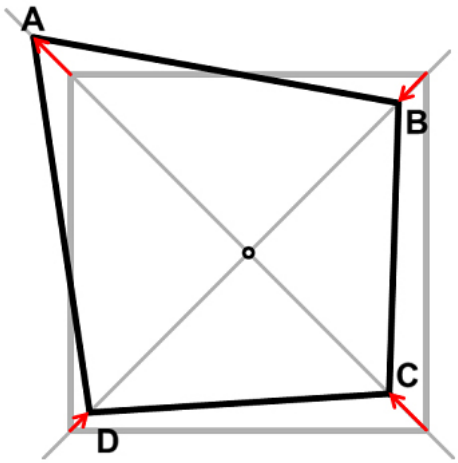


Figura 5.5: Modificación inicial de los vértices para el algoritmo fractal.

A continuación el algoritmo obtiene un punto aleatorio P de uno de los segmentos resultantes, tal y como se muestra en la figura 5.6.

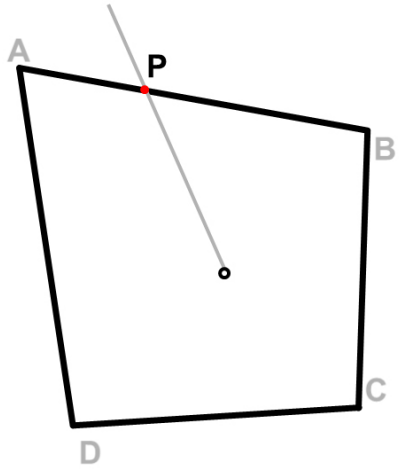


Figura 5.6: Selección de un punto aleatorio en el algoritmo fractal.

Este punto se desplaza a lo largo de la línea que pasa por dicho punto y por el centro del cuadrado original (figura 5.7).

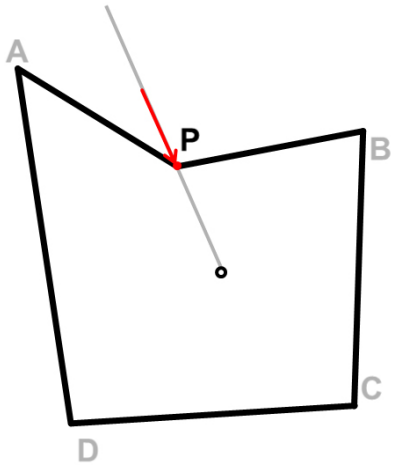


Figura 5.7: Modificación de segmento en el algoritmo fractal.

El algoritmo repite los dos últimos pasos en todos los segmentos de manera recursiva y finalmente resulta la forma relativa de la veta (figura 5.8).

La distancia de desplazamiento de los puntos se calcula mediante la ecuación 5.F, donde i es la profundidad de la recursividad del algoritmo fractal; d_i es el desplazamiento del punto en una profundidad i del mismo algoritmo (d_0 es una constante), siendo d_i el máximo y $-d_i$ el mínimo valor de desplazamiento; y $SmoothConstant$ es una constante que indica la pérdida de rugosidad de cada iteración. Los valores para

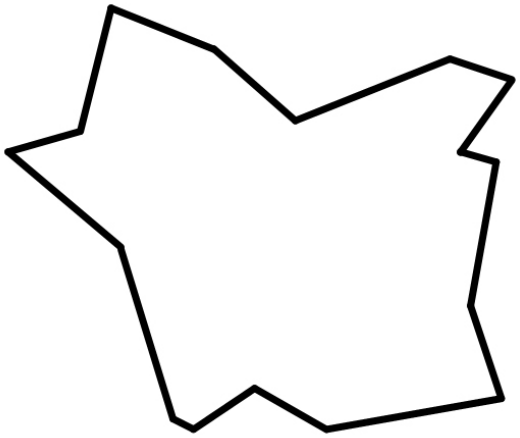


Figura 5.8: Resultado del algoritmo fractal.

las dos constantes se obtienen de los parámetros de rugosidad de los materiales.

Por causas de rendimiento, se ha limitado la profundidad del algoritmo recursivo que define el fractal a 5.

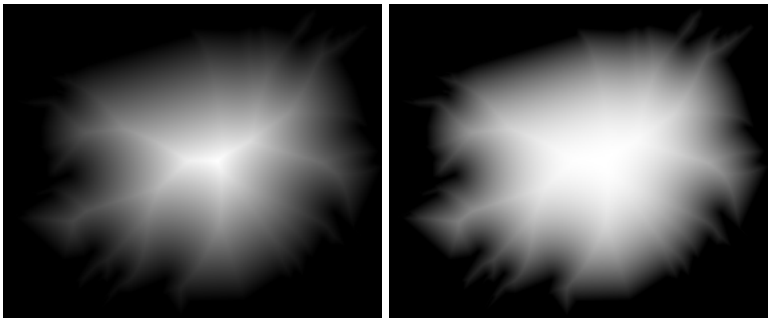
$$d_i = \frac{d_{i-1}}{SmoothConstant} \tag{5.F}$$

Para finalizar la construcción de la forma de la veta se redimensiona y se sitúa la forma relativa de la veta obtenida en este punto hasta que toque los cuatro lados del *bounding square* final sin salirse. De esta manera, tanto la forma como la posición final de la veta quedan establecidas.

Paso 6: Generación de las alturas de la veta

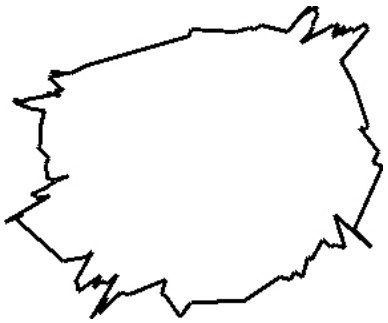
A continuación se construyen las alturas de la veta. Este proceso está basado en la distancia entre cada punto y el punto más cercano de la forma fractal. Tanto la distancia entre los puntos y la forma fractal, como la altura de la veta, se encuentran normalizados (valores dentro del rango $[0,1]$).

El cálculo de las alturas se realiza mediante la ecuación 5.G, donde h_{xy} es la altura en el punto $[x,y]$, y d_{xy} es la distancia más corta entre el punto $[x,y]$ y el borde de la forma fractal. Esta ecuación está basada en la ecuación de la circunferencia, aunque sitúa el centro de esta en el centro de la forma fractal para obtener la altura máxima en los puntos cuya distancia hasta el borde sea máxima.



(a) Altura establecida directamente como la distancia hasta el borde.

(b) Altura calculada con nuestro método.



(c) Forma fractal de la veta.

Figura 5.9: Alturas de una veta.

$$h_{xy} = |\sqrt{1-(d_{xy}-1)^2}| \tag{5.G}$$

La figura 5.9 muestra las alturas de una veta calculadas con dos métodos diferentes. Por un lado, la subfigura 5.9(a) establece la altura de cada punto directamente como la distancia desde dicho punto hasta el punto más cercano de la forma fractal. Por otro lado, la subfigura 5.9(b) muestra las alturas de una veta calculadas con nuestro método (ecuación 5.G). Podemos observar que los picos del segundo método tienen formas más redondeadas y menos pronunciadas que los del primero.

Paso 7: Generación del flujo de material de la veta

Finalmente, la generación del flujo de material de la veta se realiza exactamente igual que en la generación de una capa.

5.4.3 Conversión de los datos obtenidos a nuestro sistema de representación

Una vez llegados a este punto, el terreno se encuentra representado como una colección ordenada de capas y vetas de mineral, por lo que hay que convertirlo al sistema de representación explicado en el capítulo 4. Debido a las características de nuestro método de representación, esta conversión es trivial, ya que vale con ir colocando los límites allí donde hay una capa o una veta de mineral.

El único paso de este proceso donde es necesario un cálculo algo más complejo es al determinar aquellas zonas entre capas donde los materiales se mezclan entre ellos. Para ello, se multiplica el grosor de cada capa y veta por su facilidad para mezclarse y se coloca a la altura resultante el límite que indica donde se acaba la interpolación y comienza la capa en sí misma.

5.4.4 Ejecución *online*

Anteriormente se ha explicado cómo se genera un terreno de manera *offline* (es decir, en un proceso independiente previo a la interacción). En la generación *online*, el método debe extender un terreno previamente generado mientras el usuario va explorándolo. Con el método que proponemos estos dos procesos son similares, exceptuando el hecho de que la composición de las capas y sus propiedades están previamente calculadas, ya que es necesaria solamente su extensión a una nueva porción de terreno.

Antes de nada, el método debe determinar el número de vetas de mineral a generar en la nueva parcela de terreno. Este valor se obtiene aleatoriamente entre $0,5$ y $1,5$ veces el número de vetas que contiene la porción original del terreno. Debido a que este cálculo siempre se realiza con el número de vetas que contiene la porción original del terreno (calculado mediante el parámetro densidad de vetas de los parámetros de entrada), la densidad de vetas se mantiene en todas las porciones de terreno que se generan posteriormente.

Una vez obtenido el número de vetas a generar, el método determina su posición entre las capas de una manera completamente aleatoria. El resultado de este proceso es una pila de capas y vetas.

A continuación, el método genera los elementos de la pila comenzando por la parte inferior del terreno. Para generar las capas, estas se extienden en el plano horizontal (es decir, a lo largo del eje x y el eje z) desde las porciones aledañas de terreno y se obtienen nuevas alturas con el generador de ruido Perlin, por lo que la continuidad en su forma

está asegurada. Este proceso es el mismo que el proceso para generar capas desde cero, pero la composición y propiedades de la capa ya vienen calculadas de la porción de terreno inicial. Por otro lado, la generación de las vetas se realiza exactamente igual que cuando se crean en un terreno generado desde cero.

5.5 Resultados

Esta sección presenta los resultados obtenidos por el generador en una serie de experimentos que miden la velocidad de ejecución. Además, presenta algunos ejemplos de la diversidad de terrenos que es capaz de obtener y realiza una comparación visual de los resultados con terrenos reales.

Más adelante, en el capítulo 7, realizamos un análisis de estos resultados desde un punto de vista cualitativo.

5.5.1 Velocidad de generación

Para medir la velocidad de generación hemos definido cuatro configuraciones diferentes que intentan representar el mayor abanico posible de datos de entrada (ver la tabla 5.2). Como podemos observar, se combinan dos configuraciones diferentes de materiales (**A**, una configuración simple, y **B**, más compleja) y dos configuraciones diferentes de vetas (**0**, en el cual no hay materiales para vetas, y **1**, donde sí los hay). El valor que aparece en la tabla junto al número de materiales de vetas indica la densidad de vetas en el terreno, definida en los parámetros de entrada.

	0	**1**
A	3 normales 0 *vein*	3 normales 1 *vein*(0,5)
B	6 normales 0 *vein*	6 normales 2 *vein*(0,5)

Tabla 5.2: Configuraciones evaluadas.

Para ello se han generado 100 terrenos por cada configuración de 5×5×5 *chunks* con un tamaño de 10×10×10 *voxels* cada *chunk*.[1] La tabla 5.3 muestra el tiempo medio, la desviación estándar y la desviación media, tanto de la generación *offline* como de la

[1] El método ha sido evaluado en una CPU Intel©Corerm i7 950, con una frecuencia de reloj de 3,07GHz.

extensión de manera *online* de un terreno ya creado (la nueva porción de terreno generada tiene las mismas características y dimensiones que el terreno creado de manera *offline*). Además, muestra los tiempos obtenidos si creásemos un terreno de las mismas dimensiones con un generador de ruido Perlin tradicional.

	Offline			***Online***		
	Tiempo medio (ms)	**Desv. estándar**	**Desv. media**	**Tiempo medio (ms)**	**Desv. estándar**	**Desv. media**
A0	319,14	41,94	33,16	310,59	32,20	23,64
A1	347,01	61,66	48,63	329,70	55,76	43,11
B0	324,36	46,85	34,30	309,24	32,29	25,42
B1	354,25	70,23	50,29	315,28	47,86	37,69
PN	61,25	14,46	10,12	61,25	14,46	10,12

Tabla 5.3: Resultados de tiempo.

Podemos ver el número medio de elementos generados en cada caso en la tabla 5.4.

	Elementos	
	Nº medio de capas	**Nº medio de vetas**
A0	4,65	0,00
A1	4,72	1,74
B0	4,75	0,00
B1	4,88	1,71

Tabla 5.4: Elementos generados.

Las figuras 5.10 y 5.10 muestran una comparación visual de los resultados entre diferentes configuraciones para la generación *offline* y *online*. Podemos comprobar que en los dos métodos la generación de vetas penaliza el rendimiento del proceso. A pesar de ello, los tiempos siguen siendo suficientemente bajos para la ejecución *online* del procedimiento. Además, podemos comprobar que el aumento de la complejidad en la definición de los materiales no implica un aumento considerable del tiempo de ejecución.

5.5.2 Ejemplos

La figura 5.12 muestra algunos ejemplos de terrenos obtenidos con nuestro método.

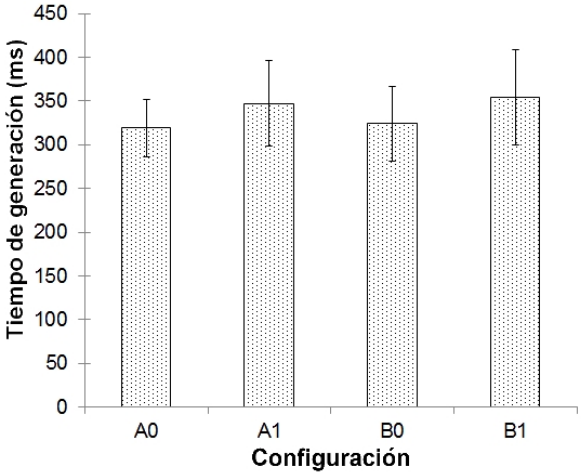


Figura 5.10: Tiempos *offline* para las diferentes configuraciones.

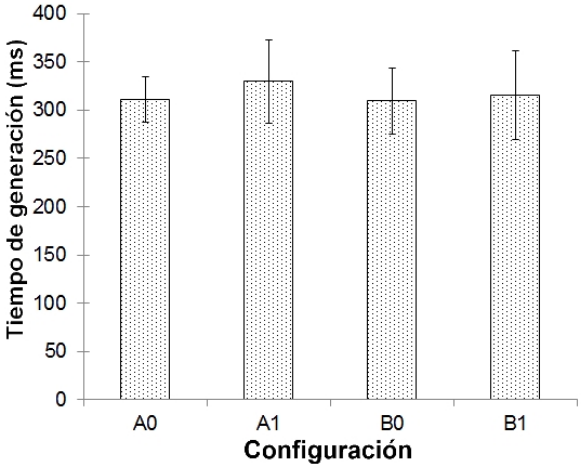
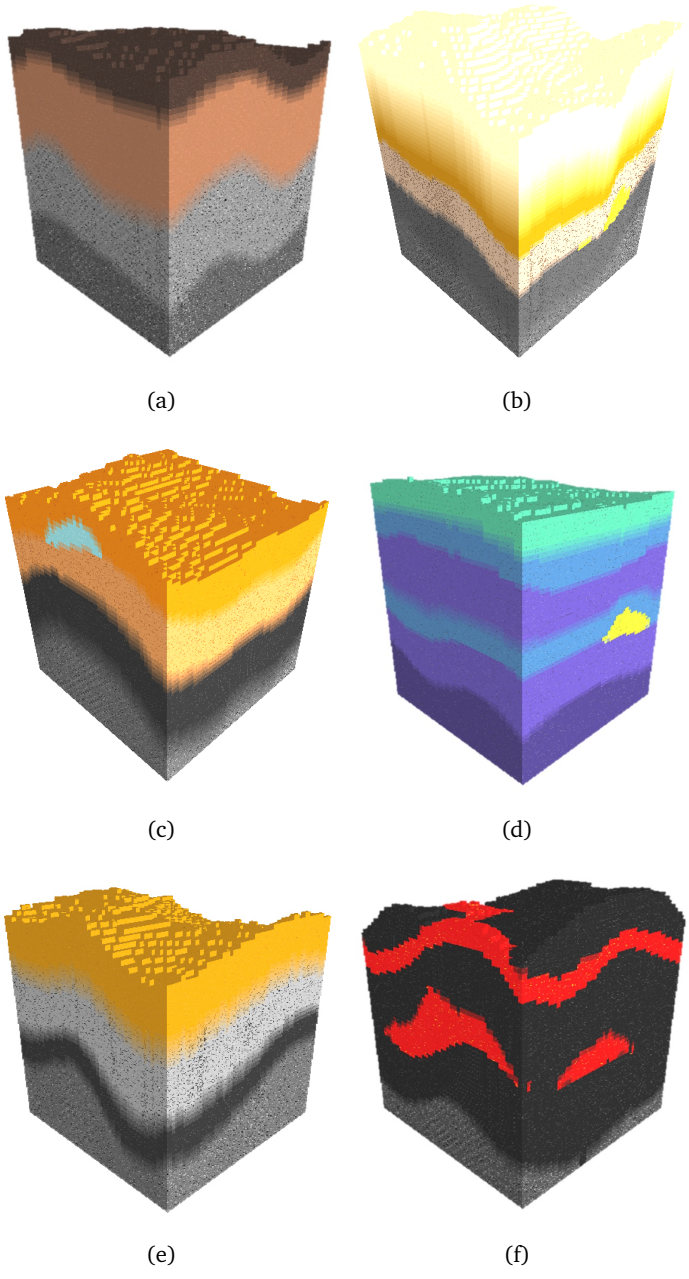


Figura 5.11: Tiempos *online* para las diferentes configuraciones.

Podemos observar como mediante modificaciones en los parámetros de entrada es posible obtener múltiples tipos de terrenos, como terrenos más tradicionales (subfiguras 5.12(a) y 5.12(e)), desiertos (5.12(b) y 5.12(c)), terrenos fantásticos (5.12(d)) o terrenos volcánicos (5.12(f)).

Además, esta figura muestra también como se estructuran y relacionan las capas entre sí y con las vetas de mineral.



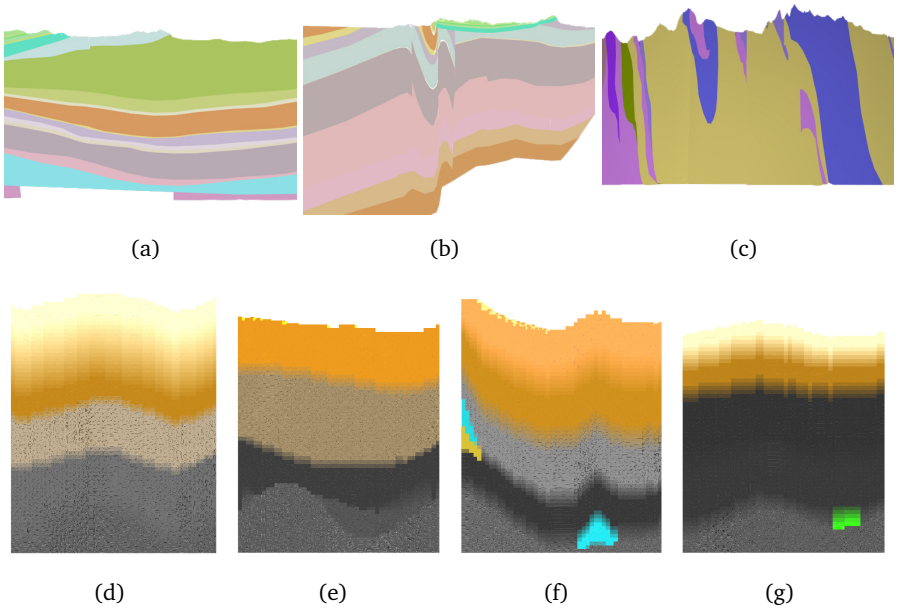
(a) (b) (c) (d) (e) (f)

Figura 5.12: Terrenos generados con nuestro método.

5.5.3 **Realismo**

La medición del realismo de un terreno generado es un proceso difícil de llevar a cabo objetivamente. Atendiendo a las métricas que proponemos en el capítulo 3 los resulta-

dos deben estar basados en nuestra realidad, pero sin llegar a ser una simulación (nivel de realismo **2**).



(a) (b) (c)

(d) (e) (f) (g)

Figura 5.13: Comparación visual con terrenos reales.

En la figura 5.13 podemos observar una comparación visual entre cortes de terreno reales[1] (subfiguras 5.13(a), 5.13(b) y 5.13(c)) y terrenos obtenidos con nuestro método (subfiguras 5.13(d), 5.13(e), 5.13(f) y 5.13(g)). Por un lado, en la subfigura 5.13(c) se muestran algunos tipos de estructuras reales que no pueden obtenerse con nuestro sistema. Por otro, podemos comprobar que tanto los terrenos reales como los artificiales están basados en capas y que existe una similitud entre ellas. Además, en ambos casos, podemos ver como las imperfecciones en el terreno afectan y se propagan verticalmente en todas las capas (ver subfiguras 5.13(b) y 5.13(f)).

[1]Contains British Geological Survey materials ©NERC 2012

«Te encuentras en un laberinto de pequeños pasillos retorcidos, todos iguales.»

Narrador (*Colossal Cave Adventure*, William Crowther y Don Woods, 1976)

Capítulo

6

Generación de elementos jugables

La generación de elementos jugables es uno de los puntos críticos en los generadores de mundos virtuales no supervisados, debido a que sus resultados deben ser directamente utilizables en un entorno interactivo con el usuario final. En este capítulo analizamos qué tipos de estructuras queremos crear y proponemos una serie de generadores para ellas.

Este capítulo se estructura de la siguiente manera: en la sección 6.1 analizamos la capa jugable y describimos y comparamos los diferentes elementos que la componen habitualmente; y en las secciones 6.2 y 6.3 describimos los métodos propuestos por nuestra investigación para generarlas y presentamos los resultados obtenidos en una serie de experimentos.

6.1 Capa jugable

La capa jugable de un mundo virtual comprende todos aquellos elementos con los que el jugador puede interactuar para que la acción del videojuego continúe, y es la principal encargada de hacer que el mundo virtual sea interesante para el jugador.

Debido a la heterogeneidad existente entre todos los géneros de videojuegos, vamos a limitar la generación de la capa jugable a aquellos juegos que basan su jugabilidad en el objetivo de esta investigación, es decir, en la exploración del mundo virtual en el que se desarrollan. Este conjunto de videojuegos abarca un gran número de géneros diferentes, como juegos de aventuras, de exploración, de plataformas o de rol. Además, debido a las características volumétricas de los mundos virtuales de esta investigación, vamos a limitar la capa jugable que se encuentra en el interior de los propios terrenos, obviando elementos de la superficie tales como ciudades, castillos u otras estructuras similares.

El conjunto de elementos jugables que cumplen con estos requisitos está compuesto, principalmente, por dos elementos: sistemas naturales de cuevas y mazmorras manufacturadas.

A primera vista podemos suponer que estos dos elementos son realmente el mismo, aunque un análisis más exhaustivo nos presenta unas diferencias importantes, resumidas en la tabla 6.1.

6.1.1 **Sistemas de cuevas**

Los sistemas de cuevas naturales representan galerías subterráneas en el terreno y presentan las siguientes características:

- Estructura laberíntica con múltiples cruces de galerías donde es fácil desorientarse.
- No tienen definido un principio, un final ni una estructura interna específica, si no que presentan una estructura con características aleatorias.
- Irregularidades en las paredes, suelos y techos de la cueva, que simulan que la cueva ha sido construida mediante un proceso natural.
- Diferentes tipos de encuentros eventuales, como pueden ser cámaras de tesoro o madrigueras de enemigos.

Es decir, los sistemas de cuevas son entornos de una extensión indeterminada, con una apariencia irregular y con una estructura donde prima la aleatoriedad. Además, la jugabilidad que presentan se basa en la exploración sin un objetivo concreto, y recompensa al jugador con diferentes encuentros a lo largo de dicha exploración.

6.1.2 **Mazmorras**

Las mazmorras manufacturadas representan estructuras como minas, mazmorras de castillos o ciudades subterráneas, y presentan las siguientes características:

- Estructura definida con habitaciones, pasillos, corredores, escaleras, etc.
- Tienen definida una o varias entradas determinadas, así como un final.
- Al intentar simular estructuras manufacturadas por seres inteligentes las paredes, techos y suelos son, en su mayor parte, regulares.
- Los encuentros y retos que presentan en su interior se encuentran estructurados (por ejemplo, primero se encuentra la llave en una habitación y, posteriormente, la puerta que abre).
- Tienen un objetivo global, como puede ser derrotar a un villano, salvar a una princesa o encontrar un tesoro.
- Están divididos en zonas con objetivos locales, como abrir una puerta, solucionar un puzle, derrotar a un enemigo menor o sortear una trampa.

Visto esto, podemos decir que las mazmorras son entornos limitados con una estructura definida. Además, presentan una jugabilidad orientada a la consecución de un objetivo global mediante la superación de retos locales.

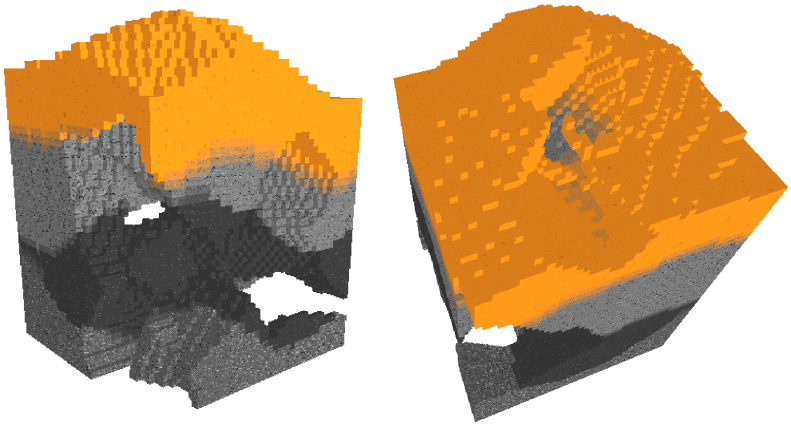
Una característica que se ha observado en este tipo de elementos es que se pueden clasificar por familias dentro de cada juego. Es decir, existen conjuntos de mazmorras con características y objetivos similares pero con breves diferencias entre ellas, como la existencia de alguna habitación, pasillo o estructura diferente.

	Cuevas	**Mazmorras**
Estructura	Aleatoria	Definida
Objetivo	No definido	Específico
Apariencia	Irregular	Regular
Jugabilidad	Exploración	Puzles y retos concretos
Extensión	Indeterminada	Determinada

Tabla 6.1: Principales diferencias entre sistemas de cuevas naturales y mazmorras manufacturadas.

6.2 Generación de cuevas

En esta sección profundizamos en el método de generación utilizado para crear cuevas (de las que podemos ver un ejemplo en la figura 6.1), así como en sus características principales. Además de la propia explicación paso a paso del algoritmo, presentamos tanto los principios matemáticos en los cuales se basa como los datos de entrada del algoritmo. Finalmente, mostramos los resultados de nuestro método en un conjunto de experimentos diseñados para medir su eficiencia.



(a) (b)

Figura 6.1: Cuevas de ejemplo generadas con nuestro método.

6.2.1 Características principales

Las características del generador de cuevas, al igual que las características del generador básico de terreno o del sistema de representación, se encuentran influenciadas por los requisitos establecidos en el capítulo 3, y son las siguientes:

6.2.1.1 Cuevas basadas en puntos de interés

La construcción de cuevas está basada en puntos de interés.[1] Estos puntos, definidos por el diseñador, representan cámaras subterráneas con un interés jugable especial, como pueden ser zonas de encuentros con enemigos, entradas al sistema de cuevas, zonas de tesoro o cualquier zona que se le ocurra al diseñador. Este tipo de construcción contribuye a aumentar tanto el interés jugable del resultado como la estructura de la cueva.

Estos POIs, como vemos más adelante en la sección 6.2.3, se relacionan entre ellos mediante afinidades no restrictivas, obteniendo una jugabilidad abierta, donde pueden aparecer diferentes elementos inesperados en la cueva.

6.2.1.2 El resultado no es predecible

Debido a las características del método utilizado, el sistema de cuevas resultante no es predecible, ya que es común la generación de elementos inesperados por el diseñador, contribuyendo al factor innovador del algoritmo.

6.2.1.3 Resultados basados en la jugabilidad

Los sistemas de cuevas resultantes están basados en la jugabilidad definida por el diseñador mediante los POIs. A pesar de ello, como vemos más adelante en la sección 6.2.3, los POIs se relacionan entre ellos mediante simples afinidades, por lo que la estructura resultante es muy poco restrictiva.

[1] De aquí en adelante haremos referencia al concepto «punto de interés» mediante sus siglas en inglés POI.

6.2.1.4 **Parámetros de entrada relacionados exclusivamente con el resultado**

Como vemos más adelante, todos los parámetros de entrada se encuentran directamente relacionados con el sistema de cuevas y con la jugabilidad que el diseñador desea otorgarle, lo que evita, de esta manera, conceptos técnicos relacionados con el método matemático utilizado. Esta característica contribuye a mejorar la usabilidad global del generador procedural.

6.2.1.5 **Proceso no interactivo**

El método de generación es completamente autónomo; requiere únicamente una entrada inicial de parámetros. Si el generador no tuviese esta característica, solamente podría ejecutarse de una manera *offline*.

6.2.1.6 **Ejecución híbrida**

El método se ejecuta de una manera híbrida, esto es, aunque puede ejecutarse de manera independiente en el sistema final, no puede ejecutarse completamente *online* debido a la cantidad tiempo que tardan en llevarse a cabo algunos de sus pasos.

6.2.1.7 **Conocimiento sobre el resultado en los parámetros de entrada exclusivamente**

El método por sí mismo no contiene ningún tipo de conocimiento. En nuestra propuesta, toda la información relativa al resultado se provee en los datos de entrada, lo que aumenta tanto el control como la ampliabilidad del sistema.

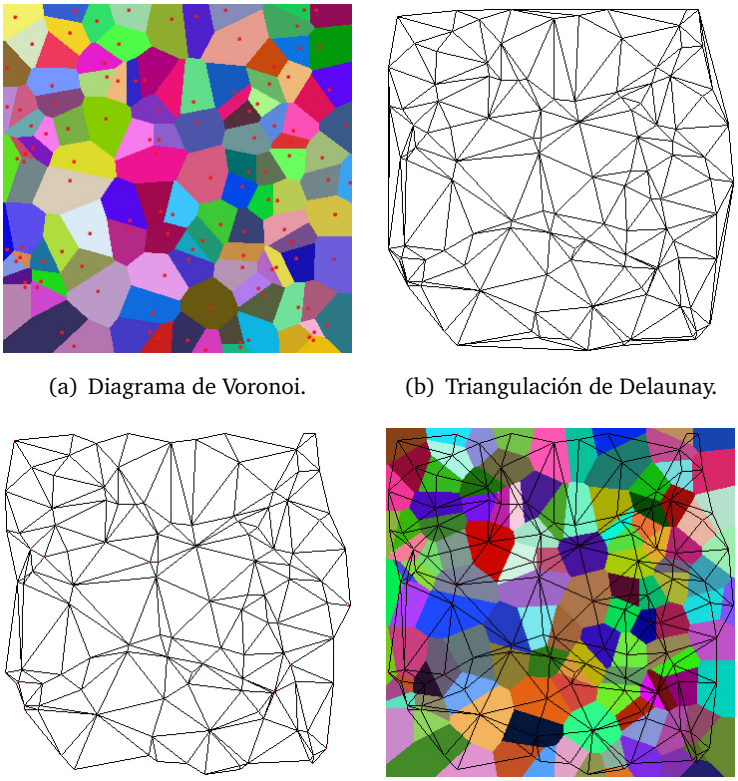
6.2.1.8 **Escala relativa**

La escala del resultado se determina mediante un parámetro en los datos de entrada, lo que permite al diseñador determinar cualquier nivel de escala para la cueva. De esta manera, la escalabilidad del método está asegurada.

6.2.2 **Materiales y métodos**

El método utilizado para la generación de cuevas se basa en tres elementos: los diagramas de Voronoi para determinar la forma de las cámaras subterráneas, la triangulación

de Delaunay para relacionar las diferentes cámaras subterráneas en un grafo y los algoritmos de búsqueda de caminos para determinar las galerías que unen dichas cámaras.



(a) Diagrama de Voronoi.

(b) Triangulación de Delaunay.

(c) Triangulación de Delaunay modificada por nuestro método.

(d) Relación entre el diagrama de Voronoy y la triangulación modificada.

Figura 6.2: Diagrama de Voronoi, triangulación de Delaunay de sus semillas, triangulación modificada por nuestro método y su relación. La triangulación de Delaunay se modifica mediante la eliminación de las aristas que unen nodos cuyas celdas de Voronoi son vecinas fuera del espacio limitado por el rectángulo que delimita la imagen.

6.2.2.1 **Diagrama de Voronoi**

Los diagramas de Voronoi [Vor08] establecen un método para dividir el espacio en regiones (llamadas celdas de Voronoi). La división se basa en una serie de puntos (llamados semillas) repartidos por el espacio. Podemos ver un ejemplo de un diagrama de Voronoi en dos dimensiones en la subfigura 6.2(a).

La división se realiza asignando a cada semilla su celda correspondiente, tal que to-

dos los puntos pertenecientes a la celda se encuentran más cerca de la semilla asignada a dicha celda que a cualquier otra semilla.

Los diagramas de Voronoi han sido tradicionalmente utilizados en áreas tanto científicas como técnicas para múltiples propósitos [Aur91], incluyendo entre ellos la generación de contenido procedural [Ols04].

6.2.2.2 **Triangulación de Delaunay**

La triangulación de Delaunay [Del34] es una técnica matemática para triangular el espacio utilizando un conjunto de puntos como punto de partida. En dicha triangulación, no existe ningún punto dentro de la circunferencia circunscrita de ningún triángulo. Esta técnica ha sido ampliamente utilizada para la generación de mayas tridimensionales [BE92]. Podemos ver un ejemplo de una triangulación de Delaunay en dos dimensiones en la subfigura 6.2(b).

La triangulación de Delaunay corresponde con el gráfico dual del diagrama de Voronoi, es decir, los centros de las circunferencias circunscritas de los triángulos obtenidos en la triangulación son los vértices de las celdas del diagrama de Voronoi. La consecuencia de esta característica es que las celdas asignadas a cualquier pareja de puntos conectados en la triangulación de Delaunay son colindantes entre ellas. La subfigura 6.2(d) muestra visualmente esta relación.

Una característica especial de la triangulación de Delaunay en nuestro método es que se modifica para que el grafo resultante enlace solo aquellas parejas de nodos cuyas celdas de Voronoi son vecinas dentro de la región del terreno. Podemos ver una triangulación modificada de esta manera en la subfigura 6.2(c) (en este caso, la región del terreno está delimitada mediante el rectángulo que define la imagen).

6.2.2.3 **Algoritmos de búsqueda de caminos**

Los algoritmos de búsqueda de caminos son algoritmos encargados del proceso de búsqueda del camino más corto entre dos puntos en el espacio. Existen muchos algoritmos diferentes y nuestro método puede adaptarse para utilizar cualquiera de ellos. En nuestro caso hemos utilizado el algoritmo A*[1] debido a que siempre encuentra el camino óptimo desde el punto inicial hasta el punto final y es uno de los algoritmos de búsqueda de caminos más utilizados en videojuegos.

[1] pronunciado A *star* o A estrella

ID	Nombre	Posición	Profund.	Parametros especiales	Tamaño	Ramif.
0	Entrada	*surface*		*start(1)*	0,5/1/1	1/1
1	Encuentro	*inside*	0/-/1	*none*	1/ 1,5/2	1/3
2	Tesoro	*inside*	0/0,5/1	*none*	0,5/0,5/2	0/3
3	Ruta	*inside*	0/-/1	*union(0,5)*	1/1/1	0/3
4	Salida	*surface*		*none*	0,5/1/1	0/0

Tabla 6.2: Ejemplo de conjunto de descriptores de tipos POIs

6.2.3 Datos de entrada

Los datos de entrada se estructuran en tres grupos principales de parámetros: descripción de los tipos de POI, descripción de las relaciones entre los tipos de POI y parámetros globales.

6.2.3.1 Descriptores de los tipos de POI

Inicialmente se describen los diferentes tipos de POI. Por cada tipo de POI existen los siguientes parámetros:

- **ID:** Identificador numérico único del tipo de POI.
- **Nombre:** Nombre del tipo de POI.
- **Posición:** Posición en el terreno en la que se desea que se encuentren los POIs de este tipo. Puede definirse como *inside* (toda la cámara subterránea se encuentra en el interior del terreno) o *surface* (el POI se encuentra en la superficie, siendo una entrada al sistema de cuevas).
- **Profundidad:** Profundidad a la cual van a ubicarse los POIs de este tipo. Este parámetro solamente se tiene en cuenta para los tipos de POI definidos como *inside*. Está formado por tres valores: profundidad relativa mínima, profundidad relativa óptima y profundidad relativa máxima.
- **Parámetros especiales:** Lista de parámetros para el tipo de POI. Algunos de estos parámetros tienen un subparámetro asociado:

  start(N) Determina que este tipo de POI es un punto inicial de la cueva. *N* indica la probabilidad de que una cueva comience por este tipo de POI o por

otro que también tenga este parámetro definido.

union(N) Este parámetro indica que este tipo de POI puede alcanzarse desde más de un POI anterior, es decir, que si se crean dos POIs de este mismo tipo, existe una posibilidad (dada por N) de que se unan en un mismo punto, caso en el cual se obtendría un solo POI en vez de dos.

- **Tamaño:** Tamaño relativo de la cámara subterránea. Está formado por tres valores: tamaño relativo mínimo, tamaño relativo óptimo y tamaño relativo máximo.
- **Ramificaciones:** Número de POIs que van a construirse inmediatamente después de este un POI de este tipo, y que generan una división de caminos en la cueva. Está formado por dos valores: número mínimo de ramificaciones y número máximo de ramificaciones.

La tabla 6.2 presenta un ejemplo de un grupo de descripciones de tipos de POI. Un valor óptimo con - indica que ese elemento sigue una distribución de ruido blanco y que solamente se tienen en cuenta los valores mínimo y máximo.

Tipos de POI Ant./Sig.	Cruzable	Afinidad	Nº de canales	Sinuosidad	Distancia
1 / 0	-	0	-	-	-
1 / 1	true	0,01	1/3	0,1/1	0/0,5/1
1 / 2	true	1	1/1	0/0,5	0/0,1/0,2
1 / 3	true	0,5	1/3	0/1	0/0,5/1
1 / 4	true	0,1	1/3	0/1	0/0,5/1

Tabla 6.3: Ejemplo de las interrelaciones de un tipo de POI.

6.2.3.2 Descriptores de las relaciones entre tipos de POI

A continuación, se describen las relaciones entre los diferentes tipos de POI. Por cada pareja de tipos de POI se definen los siguientes parámetros:

- **POI previo:** ID del primer tipo de POI de la relación.
- **POI siguiente:** ID del segundo tipo de POI de la relación.
- **Camino cruzable:** Parámetro booleano que indica si el camino que une los dos tipos de POI de la relación puede ser cruzado por otros caminos de otras relaciones. Este parámetro debería ser falso cuando se requiere que dos POIs estén

siempre uno después del otro y no pueda accederse al segundo sin pasar por el primero (por ejemplo, un tesoro grande después de un monstruo importante).

- **Afinidad:** Afinidad de la relación (1 indica la máxima afinidad, mientras que una 0 indica que esos dos tipos de POI son incompatibles).
- **Número de canales:** Número de caminos subterráneos que une los dos tipos de POI. Está formado por dos valores: número mínimo de caminos y número máximo de caminos.
- **Sinuosidad:** Sinuosidad relativa de los caminos que unen los dos tipos de POI. Está formada por dos valores: sinuosidad relativa mínima y sinuosidad relativa máxima.
- **Distancia:** Distancia relativa entre los dos tipos de POI de la relación. Está formada por tres valores: distancia relativa mínima, distancia relativa óptima y distancia relativa máxima.

La tabla 6.3 presenta un ejemplo de conjunto de parámetros para definir el conjunto de relaciones de un tipo de POI. Debido a la extensión de estos parámetros solamente se incluyen las relaciones de un tipo de POI con el resto (en este caso del tipo con identificador 1 de la tabla 6.2 con el resto de tipos de POI de dicha tabla). Las relaciones con afinidad 0 no tienen el resto de parámetros establecidos, debido a que indica que nunca va a ocurrir esa relación, y por tanto se obvian.

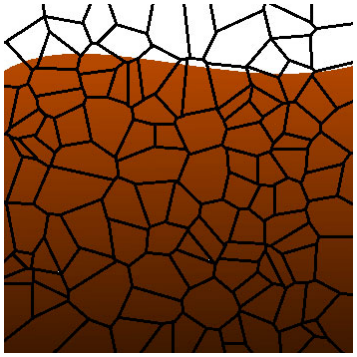
6.2.3.3 **Parámetros globales**

Finalmente, se proveen los parámetros globales, que son los siguientes:

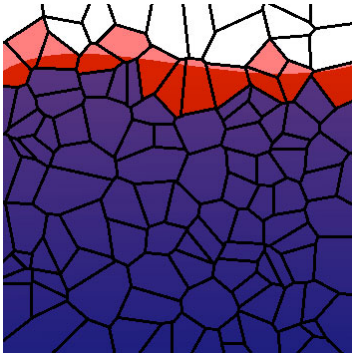
- **Profundidad:** Profundidad del terreno. Normalmente este valor se obtiene del terreno sobre el cual se está construyendo la cueva (si se desea que esta se construya teniendo en cuenta solamente una región superior del terreno es posible definir una profundidad menor).
- **Sinuosidad global:** Establece el grado de sinuosidad general para toda la cueva. Este parámetro convierte en valores absolutos los valores relativos de sinuosidad definidos en las relaciones de los tipos de POI.
- **Distancia global entre POIs:** Define la distancia entre POIs para toda la cueva. Este parámetro convierte en valores absolutos los valores relativos de distancia entre POIs definidos en las relaciones de sus tipos.

- **Escala:** Normalmente es la misma que la del terreno y puede obtenerse de él. Se establece en una unidad de medida a la que hemos llamado punto [1], y mide en esta unidad el lado de cada *voxel* del terreno.

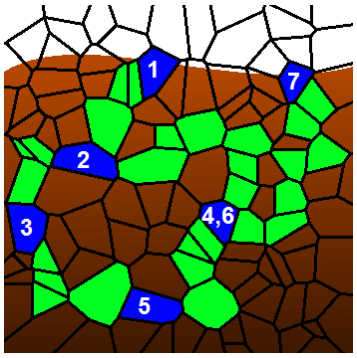
- **Tamaño global de las cámaras:** Establece el tamaño global de las cámaras subterráneas de toda la cueva. Determina el número medio de puntos cúbicos (unidad definida en el parámetro de escala) para cada cámara subterránea.



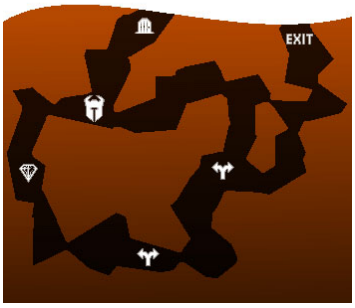
(a) Terreno bidimensional con el diagrama de Voronoi superpuesto.



(b) Catalogación de las celdas (en rojo las celdas *surface* y en azul las *inside*)



(c) Celdas seleccionadas por el algoritmo para construir la cueva (en azul los POIs, que se encuentran numerados en orden de construcción, y en verde las galerías).



(d) Cueva excavada en el terreno y con los eventos de los POIs asignados.

Figura 6.3: Pasos para la generación de cuevas. Hemos utilizado la definición de tipos de POI proporcionada en la tabla 6.2.

[1] En nuestro método, un punto es una unidad de media abstracta para medir longitud, y es el diseñador el que decide qué representa (centímetros, metros, kilómetros o cualquier otro tipo de unidad).

6.2.4 Proceso de generación

6.2.4.1 Pasos para la generación

El algoritmo se compone de los siguientes pasos:

Paso 1: Diagrama de Voronoi y triangulación de Delaunay

El primer paso para crear el sistema de cuevas es la generación del diagrama de Voronoi con su correspondiente triangulación de Delaunay.

Este proceso comienza con la generación de las semillas para el diagrama de Voronoi. Estas semillas son una serie de puntos que siguen una distribución de ruido blanco[1] en el espacio que ocupa el terreno sobre el cual se va a generar la cueva.

El número de semillas a distribuir se determina mediante la ecuación 6.A, donde N es el número de *voxels* total de la porción del terreno donde se va a construir la cueva, C es el tamaño global de las cámaras obtenido de los parámetros globales de los datos de entrada y S es la escala del terreno obtenido también de los parámetros globales de los datos de entrada.

$$Seeds = \frac{N}{C} S^3 \tag{6.A}$$

A continuación el algoritmo utiliza estas semillas para construir el diagrama de Voronoi y la triangulación de Delaunay.

La subfigura 6.3(a) muestra un terreno bidimensional de ejemplo con un diagrama de Voronoi generado sobre él.

Paso 2: Generación del primer punto

El siguiente paso consiste en la creación del primer POI de la cueva.

Primero se selecciona el tipo de POI a crear de manera aleatoria, utilizando para ello aquellos tipos con el parámetro especial *start* definido y la probabilidad del subparámetro que lo acompaña.

Después, el algoritmo selecciona la celda del diagrama de Voronoi más adecuada para posicionar este POI comparando las características de su tipo definidas en los datos

[1] El ruido blanco es una señal aleatoria cuyos valores no guardan correlación estadística alguna.

de entrada con las características de cada celda. Este proceso se explica en la sección 6.2.4.3.

Finalmente, el punto generado se almacena en una lista de puntos a procesar que contiene todos aquellos puntos de la cueva a los cuales se ha llegado y desde los cuales puede seguir creciendo.

Paso 3: Generación del resto de la cueva

Este paso consiste en la generación de la cueva en sí misma. Se ejecuta de manera iterativa: inicialmente, se obtiene el primer POI de la colección de puntos a procesar y se elimina de dicha colección y, a continuación, se construye el siguiente POI y las galerías que los unen.

Esta construcción se realiza mediante el siguiente proceso:

Primero se calcula el número de ramificaciones que parten desde el POI. Este valor se obtiene aleatoriamente entre el mínimo y el máximo de ramificaciones establecido en la definición del tipo de POI. El número de ramificaciones determina el número de POIs conectados al POI actual que a continuación se van a construir. Si este valor es igual a 0, el algoritmo considera que este punto es un camino sin salida de la cueva, por lo que obtiene el siguiente punto de la colección de puntos a procesar y repite el proceso. Si el número de ramificaciones es mayor que 0, el algoritmo construye un nuevo POI por cada ramificación.

Para construir cada nuevo POI, primero se selecciona el nuevo tipo de POI que se va a construir (proceso explicado en profundidad en la sección 6.2.4.2). A continuación, se posiciona el POI en el terreno, buscando la celda del diagrama de Voronoi que mejor satisfaga sus necesidades (este proceso se explica en la sección 6.2.4.3) y se construyen las galerías entre los dos. El número de galerías entre los dos puntos se obtiene aleatoriamente entre el mínimo y el máximo de canales definido la relación entre los dos tipos de POI definida en los parámetros de entrada. El proceso de generación de cada galería entre los dos POIs se define en profundidad en la sección 6.2.4.4.

Finalmente, cuando todas las ramificaciones se encuentran construidas, los nuevos puntos se añaden a la colección de puntos a procesar.

Este proceso finaliza cuando, al obtener el siguiente POI de la lista de puntos a procesar, esta se encuentre vacía.

La subfigura 6.3(c) muestra un conjunto de celdas seleccionadas por este método

para construir una cueva.

Paso 4: Vaciado de la cueva en el terreno

El último paso del proceso corresponde al vaciado de las celdas, obtenidas en los pasos anteriores, en el terreno sobre el cual se está construyendo la cueva.

La subfigura 6.3(d) muestra el resultado final del proceso: una cueva en el terreno con los roles de los POIs asignados.

6.2.4.2 Selección del tipo de POI

El objetivo de este proceso consiste en la selección del siguiente tipo de POI que se va a construir en la cueva. Esta decisión depende exclusivamente de las afinidades con el resto de tipos de POI del último tipo de POI generado.

Primero se normalizan todas las afinidades del último tipo de POI construido con el resto de tipos de POI. A continuación, se selecciona el siguiente tipo de POI de una manera aleatoria utilizando para ello las afinidades normalizadas previamente calculadas.

Un caso particular son aquellos tipos de POI que tienen una afinidad igual a 0 con todo el resto de tipos. En estos casos se descarta la creación de un nuevo POI, eliminando esa ramificación de la cueva.

6.2.4.3 Ubicación del POI

Este proceso es el responsable de posicionar el siguiente POI en una celda del diagrama de Voronoi generado previamente. Para ello, se necesita el tipo del POI a construir. Además, si el punto a ubicar no es el primero de la cueva, también es necesaria la ubicación y el tipo del POI inmediatamente anterior a este.

Los pasos para llevar a cabo este procedimiento difieren dependiendo de si el tipo de POI a ubicar tiene definido el parámetro *union* dentro de sus parámetros especiales o no.

Por un lado, si el parámetro *union* no está definido, este proceso se realiza mediante el diagrama de flujo que se muestra en la figura 6.4 y que tiene los siguientes pasos:

1. **Selección de candidatos:** Primero se seleccionan aquellas celdas del diagrama

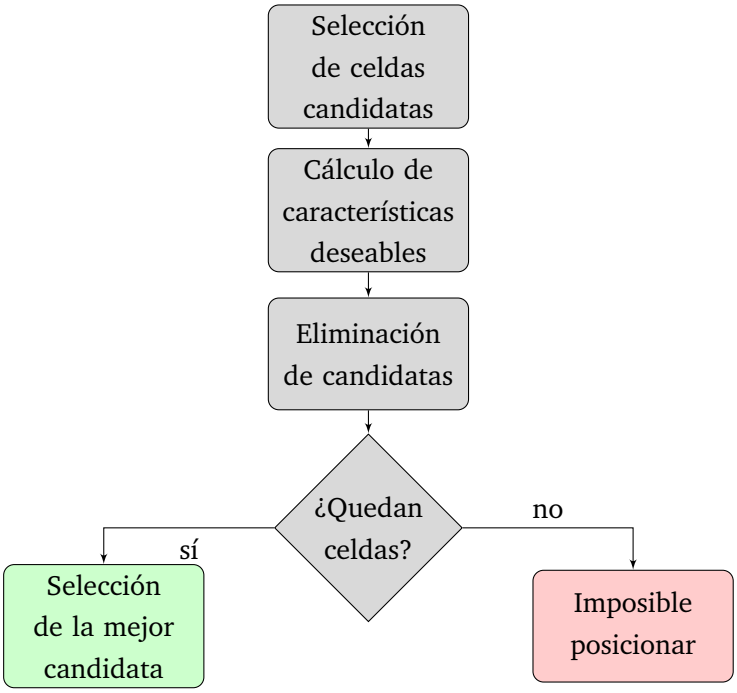


Figura 6.4: Algoritmo de posicionamiento de POIs para tipos de POI que no tengan el parámetro *union* definido.

de Voronoi que cumplan los requisitos de posición establecidos en los datos de entrada (*inside* o *surface*, podemos ver un diagrama de Voronoi con sus celdas clasificadas en la subfigura 6.3(b)). Además, las celdas ya asignadas a otros POIs, las celdas pertenecientes a caminos no cruzables (caminos entre dos POIs cuya relación ha sido definida como no cruzables en los datos de entrada) y sus paredes (es decir, las celdas vecinas en el grafo obtenido por la triangulación de Delaunay de aquellas celdas que pertenecen a un camino no cruzable) se eliminan de la lista de candidatos.

2. **Obtención de las características deseables:** A continuación se obtienen la características deseables para la celda en la que se va a ubicar finalmente el POI. Estas características son la profundidad de la celda en el terreno, el tamaño de la celda y la distancia hasta la celda en la que se ubica el POI inmediatamente anterior de la cueva (esta distancia solamente se tiene en cuenta si el POI a ubicar no es el primer punto de la cueva).

   Los valores para estas características deseables se calculan multiplicando los valores relativos definidos en los datos de entrada (la profundidad y el tamaño de la descripción del tipo del POI, y la distancia entre los puntos de la descripción

de la relación entre los dos tipos de POI) por los valores absolutos definidos en los parámetros globales (profundidad del terreno, tamaño global de las cámaras y la distancia global entre POIs).

3. **Eliminación de candidatos:** Una vez recopiladas las celdas candidatas y calculados los valores finales de las características deseables, se eliminan aquellas candidatas que queden fuera de los rangos establecidos por el valor menor y mayor de cada características deseable. Si en este paso la lista de celdas candidatas queda vacía, el método considera que no se puede ubicar este POI, por lo que descarta su construcción, dejando al POI inmediatamente anterior con una ramificación de salida menos.

4. **Obtención del mejor candidato:** Para finalizar, el proceso selecciona el mejor candidato de la lista. Para ello, compara las características de cada candidato con el valor óptimo de las características deseables previamente calculadas (si no hay un valor óptimo definido, se obvia la característica, y si ninguna de ellas tiene valor óptimo, se selecciona un candidato aleatorio).

Por otro lado, si el tipo de POI a ubicar tiene definido el parámetro *union*, el proceso varía. Este parámetro indica que el POI puede ser alcanzado desde más de un POI, es decir, que tiene una probabilidad (determinada por el subparámetro *N* que lleva asignado) de ser el punto de unión de dos caminos diferentes. Para conseguir este tipo de construcción, estos POIs se ubican en la misma celda que otros POIs del mismo tipo, obteniendo diferentes entradas a la celda desde diferentes POIs. Este proceso puede verse en el diagrama de flujo de la figura 6.5 y sigue los siguientes pasos:

1. **Probabilidad de unión:** Antes de ubicar un POI con el parámetro *union* es necesario comprobar si se supera o no la probabilidad de dicho parámetro. Para ello se obtiene un valor aleatorio entre 0 y 1, y se compara con el subparámetro *N* del parámetro *union*. Por un lado, si el valor obtenido es mayor que la probabilidad, el parámetro *union* no repercute y el POI se ubica de la misma manera que un POI sin ese parámetro definido. Por otro lado, si el valor aleatorio obtenido es menor o igual a la probabilidad, el proceso continua con el siguiente paso de esta lista.

2. **Selección de candidatos:** Una vez superada la probabilidad de que el POI sea *union*, se crea una lista de celdas candidatas para ubicarlo. Para ello, se seleccionan todas aquellas celdas en las que ya se ubica un POI del mismo tipo que se va a crear. Si no existe ninguna celda con estas características en todo el diagrama, el proceso se detiene y ubica el POI con el algoritmo explicado previamente para ubicar puntos que no tengan el parámetro *union* definido.

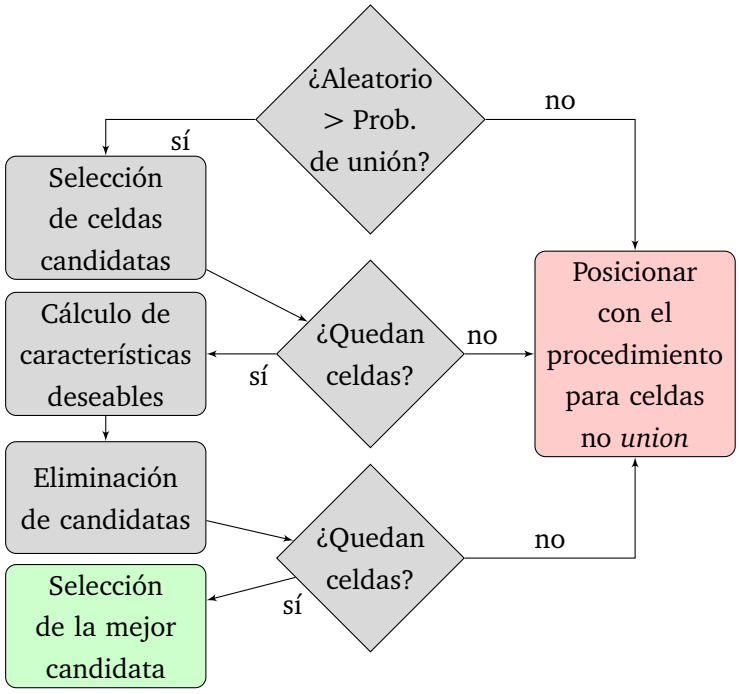


Figura 6.5: Algoritmo de posicionamiento de POIs para tipos de POI con el parámetro *union* definido.

3. **Obtención de las características deseables:** A continuación se calculan las características deseables para la celda donde se va a ubicar el POI. En este caso solamente es necesario calcular la distancia desde el punto inmediatamente anterior de la cueva.

   Este valor se calcula multiplicando la distancia relativa obtenida del descriptor de la relación entre los dos tipos de POI por el valor de la distancia global entre POIs, definido en la sección de parámetros globales de los datos de entrada.

4. **Eliminación de candidatos y obtención del mejor candidato:** El proceso termina con los dos mismos últimos pasos (3 y 4) que el procedimiento para ubicar un POI sin el parámetro *union*.

6.2.4.4 Generación del camino entre POIs

Este proceso es el procedimiento encargado de generar el camino entre dos POIs ubicados en el terreno.

El primer paso es la creación de una colección de celdas intransitables, que con-

tenga todas aquellas celdas que no pueden pertenecer al camino. Esta colección está compuesta por los siguientes grupos de celdas:

- Celdas del diagrama de Voronoi en las que ya se ha ubicado un POI.
- Celdas que pertenecen a los caminos entre dos POIs cuya relación se ha definido como no cruzable en los datos de entrada.
- Si el camino que se va a generar conecta dos POIs cuya relación se ha definido como no cruzable en los datos de entrada, todas las celdas del diagrama de Voronoi que pertenezcan a un camino definido como cruzable en los datos de entrada.
- Todas las celdas vecinas en el diagrama de Voronoi (es decir, todas aquellas que estén directamente relacionadas en el grafo obtenido por la triangulación de Delaunay) de las celdas seleccionadas en los tres puntos anteriores (es decir, todas las paredes de las celdas intransitables).

El siguiente paso es el cálculo de la sinuosidad del camino a generar. Para ello, se multiplica la sinuosidad relativa, definida en la descripción de la relación entre los dos tipos de POI que conecta el camino, por la sinuosidad global, definida en los parámetros globales.

Para finalizar, el método genera el camino entre los dos POIs utilizando un algoritmo de búsqueda de caminos sobre el grafo obtenido por la triangulación de Delaunay. Nosotros utilizamos el método A* debido a que es un método ampliamente difundido para el desarrollo de videojuegos [Nar04], aunque puede utilizarse cualquier otro. Para representar la sinuosidad previamente calculada, se añade un error al coste de movimiento de cada nodo del camino hasta el POI final. Este error se calcula aleatoriamente en el rango determinado por $-sinuosidad/2$ y $sinuosidad/2$.

Este proceso termina cuando el algoritmo de búsqueda de caminos alcance el nodo final o cuando compruebe que dicho nodo es inalcanzable. En el primer caso, el camino se obtiene y se utiliza para la generación de la cueva. En el segundo, el camino, al igual que el POI final, se descarta eliminando esa ramificación desde el POI anterior.

6.2.4.5 **Generación de varias cuevas en el mismo terreno**

Para generar varias cuevas en la misma porción de terreno el proceso es similar, aunque es posible evitar la repetición de algunos procedimientos.

Por un lado, el primer paso del proceso (generación del diagrama de Voronoi y triangulación de Delaunay) debe ejecutarse una única vez, independientemente del número de cuevas que van a generarse. Por otro lado, el resto de pasos deben ejecutarse siempre para cada cueva.

6.2.5 **Resultados**

En esta sección, además de presentar los resultados obtenidos por los experimentos realizados para medir la velocidad de ejecución del algoritmo, mostramos diferentes cuevas obtenidas por nuestro generador.

Más adelante, en el capítulo 7, realizamos un análisis de estos resultados desde un punto de vista cualitativo.

6.2.5.1 **Velocidad de generación**

Configuración	Tipos de POI	Ramificaciones de cada POI	Canales entre POIs
A0	4	1	1
A1	4	1-2	1-2
B0	8	1	1
B1	8	1-2	1-2

Tabla 6.4: Configuraciones utilizadas.

Para realizar la medición del tiempo que tarda este algoritmo en generar una cueva utilizamos cuatro configuraciones diferentes. Todas ellas tienen en común que utilizan un solo tipo de POI con el parámetro *start* definido, un solo tipo de POI con 0 ramificaciones, la misma afinidad entre todos los tipos de POI, y que todos los caminos entre diferentes tipos de POI son cruzables y tienen una sinuosidad relativa de $(0,5 - 1,5)$.

El tamaño global de las cámaras es de 100 unidades, la escala de $0,5$, la sinuosidad global de 20 y la distancia global entre POIs de 40 unidades. Todas las cuevas están construidas en una porción de terreno formada por $40 \times 40 \times 40$ *voxels* (el tamaño del terreno solo afecta al tiempo de construcción del diagrama de Vorornoi). La tabla 6.4 muestra las diferencias entre las cuatro configuraciones.

Para obtener los resultados hemos generado 200 sistemas de cuevas para cada con-

Configuración.	Número de POIs	Número de celdas de caminos	Número de celdas
A0	3,53	10,67	100
A1	5,56	20,45	100
B0	4,53	15,16	100
B0	7,41	29	100

Tabla 6.5: Elementos generados.

Configuración	Tiempo medio (ms)	Desviación estándar (ms)	Desviación media (ms)
A0	10.742,72	209,20	165,87
A1	11.213,37	960,32	491,05
B0	10.803,00	241,58	190,16
B1	11.584,60	1118,79	686,36

Tabla 6.6: Tiempo total de generación.

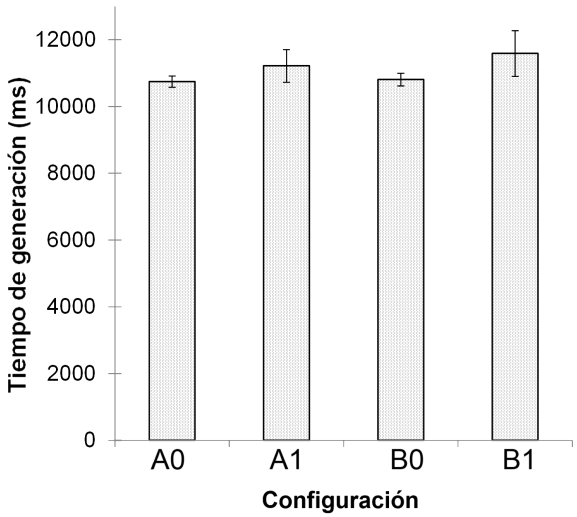


Figura 6.6: Tiempo total de generación.

figuración[1]. La tabla 6.5 muestra el número medio de POIs, el número medio de celdas pertenecientes a caminos entre POIs y el número medio de celdas generadas en los diagramas (que debido al tamaño del terreno y a la configuración de los datos de entrada

[1]Los experimentos se han realizado en un procesador Intel©Core™ i7 950, con una frecuencia de reloj de 3,07GHz.

son siempre 100).

Configuración	Tiempo medio (ms)	Desviación estándar (ms)	Desviación media (ms)
A0	10.690,72	184,93	146,45
A1	10.791,29	212,42	162,60
B0	10.700,74	208,97	161,85
B1	10.707,71	223,25	173,44

Tabla 6.7: Tiempo parcial (construcción del diagrama).

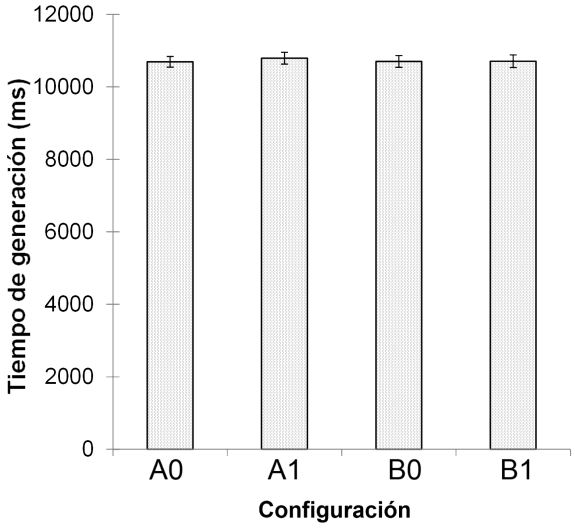


Figura 6.7: Tiempo parcial (construcción del diagrama).

La tabla 6.6 muestra los resultados de tiempo obtenidos en los experimentos. Podemos observar que nuestro método requiere más de diez segundos de media para obtener la cueva independientemente de la configuración. La figura 6.6 muestra estos resultados de manera visual.

Además de los resultados globales, hemos realizado mediciones independientes del tiempo de los dos pasos principales del proceso de generación. Por un lado, hemos medido el tiempo requerido para la generación del diagrama de Voronoi y la triangulación de Delaunay (ver tabla 6.7). Por otro lado, hemos realizado la misma medición para el proceso de generación de la cueva en sí misma (ver tabla 6.8). Como vemos, la generación de la cueva se realiza en un tiempo mucho menor que la generación de los diagramas, aunque su variabilidad es enorme.

Configuración	Tiempo medio (ms)	Desviación estándar (ms)	Desviación media (ms)
A0	52	85,54	69,67
A1	422,09	961,51	470,21
B0	102,26	120,52	106,72
B1	876,89	1118,95	682,45

Tabla 6.8: Tiempo parcial (generación de la cueva).

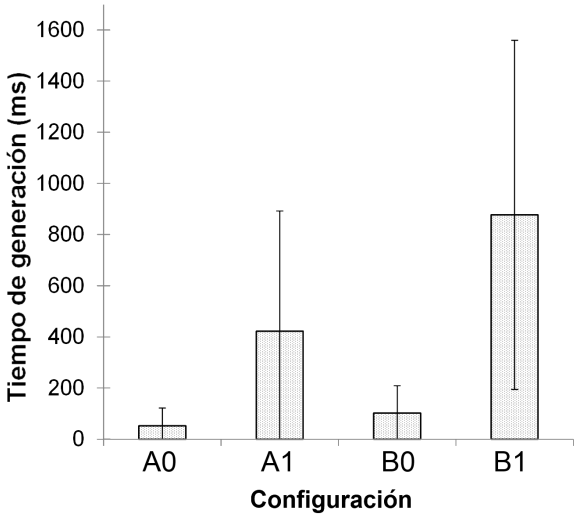


Figura 6.8: Tiempo parcial (generación de la cueva).

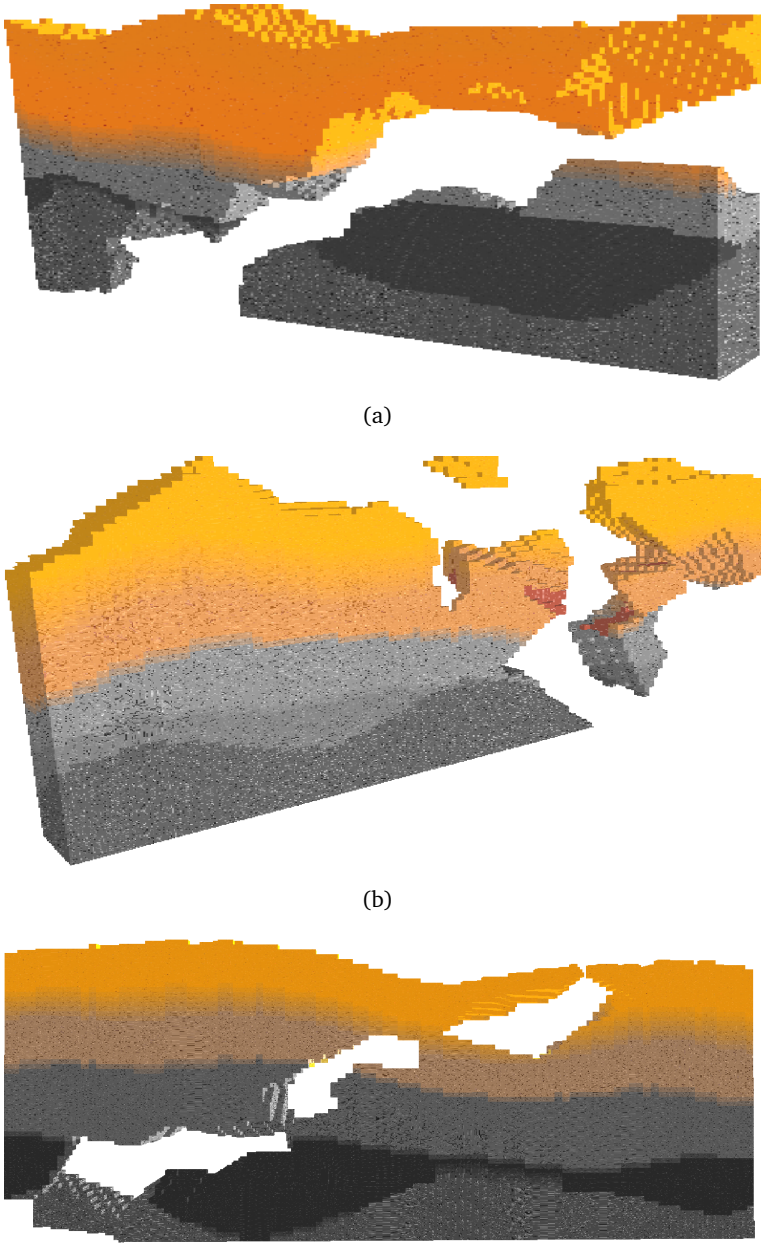
Podemos ver estos resultados de manera gráfica en las figuras 6.7 y 6.8.

6.2.5.2 **Ejemplos**

Podemos ver algunos ejemplos de cuevas generadas con nuestro método en las figuras 6.1 y 6.9.

Por un lado, la figura 6.1 muestra dos vistas diferentes de una cueva laberíntica generada con nuestro método. En la subfigura 6.1(a) podemos observar una vista general de la porción del terreno con la cueva en su interior, mientras que la 6.1(b) muestra la entrada a la cueva creada en su superficie.

Por otro lado, la figura 6.9 muestra un conjunto de cuevas sencillas creadas en cortes verticales del terreno que nos permiten apreciar la estructura de la cueva con mayor



(a)

(b)

(c)

Figura 6.9: Cuevas de ejemplo generadas con nuestro método.

facilidad. Las subfiguras 6.9(a) y 6.9(c) muestran cuevas en la que no se ha construido ninguna ramificación y donde se ha establecido una sinuosidad baja. Por el contrario, la subfigura 6.9(b) muestra una cueva con una mayor sinuosidad donde se ha obtenido, además, una ramificación.

Finalmente, en todos los ejemplos podemos observar las características irregulares de los resultados obtenidos y del aspecto natural que presentan debido a las características físicas de las celdas de los diagramas de Voronoi.

6.3 Generación de mazmorras

En esta sección detallamos el proceso de generación de mazmorras así como sus características principales, los métodos en los que está basado, los datos de entrada que requiere y los resultados obtenidos en los experimentos que miden su rendimiento.

6.3.1 Características principales

Estas características, al igual que en el resto de procesos, están basadas en los requisitos establecidos en el capítulo 3.

6.3.1.1 Definición mediante grafos

La mazmorra se define mediante grafos que determinan las zonas, las habitaciones y sus interconexiones. Además, cada nodo de estos grafos puede estar formado por otros subgrafos. Este método permite una definición de la jugabilidad de alto nivel, pero con un elevado grado de detalle que restringe la generación de la mazmorra para que se adapte a dicha definición. De esta manera, el interés de las mazmorras y su estructura se ven potenciadas, además del control por parte del diseñador.

6.3.1.2 Generación de familias de mazmorras

Los grafos de definición de la mazmorra llevan asignados una serie de condiciones que determinan la probabilidad de que un nodo o enlace exista. Además, cada nodo puede estar formado varios subgrafos, entre los cuales se selecciona uno mediante una probabilidad establecida por el diseñador.

Así, unos mismos datos de entrada permiten generar familias de mazmorras, esto es, conjuntos de mazmorras, todas ellas diferentes pero con una jugabilidad similar y con el mismo objetivo final. Por ello, cuando un terreno está poblado por mazmorras del mismo tipo, al no repetirse su estructura, las diferencias entre ellas hacen que el interés no se vea afectado. Además, como todas las mazmorras de una misma familia

comparten el mismo objetivo general, la cohesión entre ellas es fuerte, lo que potencia el nivel de estructura del mundo virtual.

6.3.1.3 **Resultado restrictivo, pero impredecible**

Como hemos visto, a pesar de que las mazmorras son más restrictivas que las cuevas debido a que se definen mediante grafos, el resultado también es impredecible, ya que tanto el proceso de situación de zonas y habitaciones como el de establecimiento de la forma final de cada habitación, incluyen una gran carga aleatoria, además de una gran influencia de su entorno. Esto ayuda a que la innovación no se vea tan afectada por el alto nivel de restricción que conllevan este tipo de estructuras.

6.3.1.4 **Resultado basado en una jugabilidad estricta**

Los grafos permiten definir la estructura de la mazmorra de una manera estricta. De este modo, es posible asignar eventos jugables (como puertas cerradas y zonas donde buscar la llave, o puzles que abren diferentes zonas de la misma mazmorra) a los diferentes nodos para obtener una jugabilidad compleja definida por el diseñador. Así, el interés se ve enormemente potenciado.

6.3.1.5 **Jugabilidad basada en objetivos multinivel**

Debido a las características del método empleado es posible definir la mazmorra con objetivos multinivel. Es decir, la mazmorra siempre va a tener un objetivo global (por ejemplo salvar a la princesa), las zonas van a tener su propio objetivo local (por ejemplo abrir la puerta que permite acceder a la siguiente zona) y finalmente las habitaciones van a presentar su reto particular (por ejemplo derrotar a un guardia o sortear una trampa).

Estos objetivos multinivel permiten que la jugabilidad de las mazmorras generadas sea similar a la jugabilidad de las mazmorras de videojuegos sin contenido procedural, potenciando el interés, la estructura, el realismo y el control del método.

6.3.1.6 **Parámetros de entrada relacionados exclusivamente con el resultado**

Al igual que en los métodos de generación anteriores, los datos de entrada se encuentran directamente relacionados con el resultado (en este caso la mazmorra), lo que

contribuye a mejorar la usabilidad global del generador procedural.

6.3.1.7 **Mazmorras multipiso**

Las mazmorras resultantes tienen en cuenta la posibilidad de generarse en múltiples pisos si es necesario, lo que requiere el posicionamiento de escaleras que conectan los diferentes pisos.

6.3.1.8 **Proceso no interactivo**

El método de generación es completamente autónomo, es decir, requiere únicamente una entrada inicial de parámetros. Esto es un requisito para que el método pueda ejecutarse de una manera *online* o híbrida (en este caso y como veremos en el siguiente punto, se ejecuta siempre de una manera híbrida).

6.3.1.9 **Ejecución híbrida**

Debido a las estrictas características de la jugabilidad de las mazmorras, la generación debe realizarse de principio a fin, imposibilitando un crecimiento de la mazmorra mientras está siendo explorada. Debido a ello, la ejecución del método se realiza de una manera híbrida y no *online*.

6.3.1.10 **Información sobre el resultado exclusivamente en los parámetros de entrada**

El método por sí mismo no contiene ningún tipo de conocimiento, ya que toda la información relativa al resultado se provee en los datos de entrada, lo que aumenta tanto el control como la ampliabilidad del sistema.

6.3.2 **Materiales y métodos**

El generador de mazmorras que proponemos a continuación se basa, principalmente, en dos conceptos: técnicas de presión para la división de estancias y algoritmos de búsqueda de caminos.

6.3.2.1 **División de estancias mediante técnicas de presión**

Como hemos visto en el estudio de la técnica, los métodos de presión se utilizan comúnmente para dividir la planta de una casa en estancias (ver sección 2.2.4.4).

El método que utilizamos en este generador está basado en las técnicas de presión utilizadas por Martin [Mar06] y Lopes *et al.* [LTS+10]. En ellas se divide el espacio en una rejilla, se posicionan las semillas de cada habitación siguiendo ciertas restricciones de adyacencia y, finalmente, se expanden en función del peso de cada habitación siguiendo unas reglas determinadas (crecimiento rectangular, crecimiento en L, etc.).

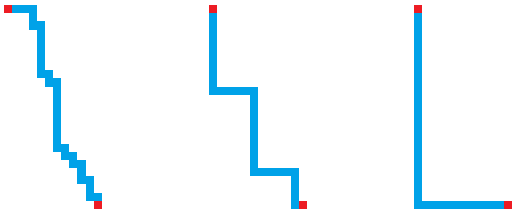
Finalmente, aseguramos la conectividad definida en el grafo de entrada generando pasillos entre las habitaciones si es necesario. Para ello, nos basamos en el método utilizado por Marson y Musse [MM10], en el que crean pasillos siguiendo paredes entre habitaciones evitando las puertas. Debido a que este acercamiento no se basa en un espacio discreto (es decir, en un espacio dividido mediante una rejilla, como en los métodos de división de estancias mediante presión), ha sido necesario adaptarlo para tener en cuenta esta característica.

6.3.2.2 **Algoritmos de búsqueda de caminos**

De la misma manera que el método para la generación de cuevas naturales visto previamente en la sección 6.2.2.3, el generador de mazmorras también se basa en los algoritmos de búsqueda de caminos.

Concretamente, el algoritmo utilizado es una versión modificada de A* para obtener caminos entre dos puntos con menos cambios de dirección que la versión tradicional del algoritmo. Para ello, en el proceso de búsqueda del camino se añade una penalización al coste del siguiente paso siempre que no continúe en la misma dirección que el paso anterior.

La figura 6.10 muestra algunos ejemplos de diferentes caminos obtenidos con el algoritmo A*. Por un lado, la subfigura 6.10(a) muestra un camino obtenido por el algoritmo tradicional donde todos los pasos tienen el mismo coste independientemente de la dirección. Por otro lado, las subfiguras 6.10(b) y 6.10(c) muestran dos caminos obtenidos con la versión modificada del algoritmo, aunque con diferentes valores asignados al coste añadido de cambiar la dirección.



(a) (b) (c)

Figura 6.10: Caminos de ejemplo obtenidos con diferentes versiones de A*. Los caminos están pintados de azul, mientras que los puntos entre los cuales se ha buscado dicho camino se encuentran pintados de rojo.

6.3.3 Datos de entrada

Los datos de entrada de nuestro generador de mazmorras se dividen en dos apartados diferenciados. Por un lado, es necesario definir los grafos que determinan la distribución de la mazmorra (se requiere un grafo principal, y las definiciones de todas las habitaciones, de todas las superzonas y de todas las zonas). Por otro lado, se establecen una serie de parámetros globales.

6.3.3.1 Descripción de los grafos utilizados

Los grafos utilizados son grafos convencionales, aunque tienen ciertas peculiaridades. Están compuestos por los siguientes elementos:

Nodos: Representan habitaciones, zonas (conjuntos relacionados de habitaciones) o superzonas (conjuntos relacionados de zonas). Todos los grafos deben tener un conjunto de nodos de entrada y otros de salida.

Enlaces: Representan la conectividad entre dos nodos (entre habitaciones, zonas o superzonas) y son direccionales (un enlace del nodo A al nodo B es diferente a un enlace del nodo B al nodo A).

Modificadores: Además de los elementos clásicos de los grafos (nodos y enlaces), nuestro método incorpora modificadores de existencia. Estos elementos definen la probabilidad de que un nodo o un enlace exista. Puede haber dos tipos diferentes de modificadores:

- Simples: Indican directamente la probabilidad de que un nodo o un enlace exista. Se definen de la siguiente manera: *A:50%*, es decir, existe una probabilidad del 50% de que el nodo A exista cuando se vaya a construir la mazmorra (cuando un nodo deja de existir también desaparecen todos sus enlaces).
- Condicionales: Indican la probabilidad de que un nodo o un enlace exista en función de una condición. Se definen de la siguiente manera: *B:A?100%:50%*, es decir, la probabilidad de que el nodo B exista es del 100% si el nodo A existe, y del 50% si el nodo A no existe. Como podemos ver, su funcionamiento es exactamente el mismo que el operador ternario existente en lenguajes como C/C++. Además, la condicional inicial permite sentencias booleanas complejas que contengan paréntesis y operadores *or* (||), *and* (&&) y *not* (!).

Los modificadores de existencia facilitan la construcción de familias de mazmorras con leves cambios entre ellas.

6.3.3.2 **Definición de habitación**

Cuando el nodo de un grafo representa una habitación, está representando una habitación real de la mazmorra. Cada habitación tiene asignado un tamaño deseado relativo que posteriormente será utilizado para el algoritmo de extensión de las habitaciones.

Además, el diseñador puede asignar un conjunto de eventos jugables a cada habitación.

6.3.3.3 **Definición de zona**

Una zona representa un conjunto de habitaciones de la mazmorra relacionadas entre sí.

Una zona se define mediante un conjunto de grafos a los que se les asigna una probabilidad, con la restricción de que la suma de las probabilidades de todos sus grafos debe ser siempre uno. De esta manera, cuando el generador construye la zona, utiliza estas probabilidades para determinar el grafo con el que se va a representar la zona. Al igual que ocurre con los modificadores de existencia, el hecho de que una zona pueda representarse por diferentes grafos, facilita la creación de familias de mazmorras con pequeños cambios entre ellas.

Para facilitar la definición de las zonas, el método permite definir tipos de zona, asignando uno de estos tipos a cada zona presente en el grafo.

Un requisito de los grafos de la zona es que todos sus nodos deben representar habitaciones.

Además, un enlace entre una zona y otro nodo, indica que todos los nodos de entrada o de salida de la zona (dependiendo de la dirección) se encuentran conectados con el otro nodo del enlace.

6.3.3.4 **Definición de superzona**

Una superzona es similar a una zona, con la excepción de que sus grafos pueden contener nodos que representan zonas u otras superzonas. Además, al contrario que una zona (que indica que sus habitaciones están próximas en el espacio), una superzona no presenta ninguna restricción de este tipo, ya que su existencia se justifica solamente porque simplifica el proceso de diseño del grafo que define la mazmorra, permitiendo repetir grupos de nodos sin tener que duplicar todos ellos (se crearía una superzona con ellos y simplemente se duplicaría dicha superzona).

Para facilitar la definición de las superzonas, el método permite definir tipos de superzonas de la misma manera que para las zonas. Así, es posible asignar uno de estos tipos a cada superzona presente en el grafo y no repetir varias veces cada definición.

Además, al igual que con las zonas, un enlace entre una superzona y otro nodo, indica que todos los nodos de entrada o de salida de la superzona (dependiendo de la dirección) se encuentran conectados con el otro nodo del enlace.

6.3.3.5 **Grafo principal**

El grafo principal define la mazmorra en su totalidad y puede contener cualquier tipo de nodo (habitaciones, zonas o superzonas).

6.3.3.6 **Parámetros generales**

Los parámetros globales se proveen como una serie de valores que definen ciertos aspectos globales del método de generación. Estos son:

- **Tamaño de las habitaciones:** Indica el tamaño global de las habitaciones de la mazmorra. Para obtener el tamaño final deseado de una habitación se multiplica este valor por el tamaño relativo de dicha habitación. Este parámetro se provee en unidades cuadradas.

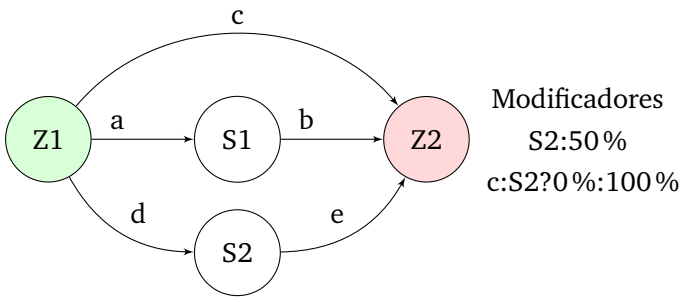


Figura 6.11: Grafo principal de ejemplo. *Z1* y *Z1* son zonas, mientras que *S1* y *S2* son superzonas. Los nodos verdes son las entradas y los rojos las salidas. Además, se incluye la tabla de modificadores del grafo.

- **Probabilidad de habitaciones en L:** Indica la probabilidad de que cada habitación tenga forma de L aunque no tenga motivo alguno.
- **Distancia entre zonas:** Distancia entre las diferentes zonas de la mazmorra.

6.3.4 Proceso de generación

El proceso de generación de la mazmorra se basa en los siguientes pasos:

Paso 1: Expansión del grafo principal

El primer paso para la generación de la mazmorra es la obtención del grafo principal expandido, obtenido mediante la sustitución de los nodos que representan superzonas por sus correspondientes grafos, resultando un grafo cuyos nodos representan únicamente zonas.

El grafo que podemos ver en la figura 6.11 es un ejemplo de grafo principal sin expandir, en el cual se han incluido los modificadores que se le aplican:

- **S2:50 %**: Indica que *S2* tiene un 50 % de posibilidades de existir.
- **c:S2?0 %:100 %**: Indica que el enlace *c* no tiene probabilidades de existencia en caso de que *S2* exista, y que sus probabilidades de existencia son del 100 % en caso de que *S2* no exista.

Además, en la figura 6.12 podemos ver el conjunto de grafos que definen la estructura de las superzonas de tipo *S* (los nodos *S1* y *S2* en el grafo principal). Existen dos

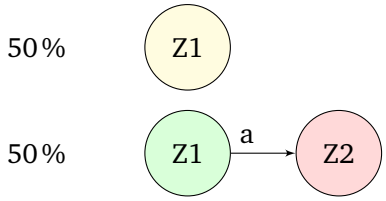


Figura 6.12: Grafos de las superzonas de tipo *S* del grafo de la figura 6.11. Está compuesta por dos subgrafos con una posibilidad de existir del 50 % cada uno. Los nodos de tipo *Z* son zonas. Los nodos verdes son las entradas, los rojos las salidas y los amarillos entradas y salidas a la vez. Los grafos de esta superzona no tienen asignado ningún modificador.

posibilidades diferentes para este tipo de superzona, cada una de ellas con un 50 % de probabilidades de ser la elegida.

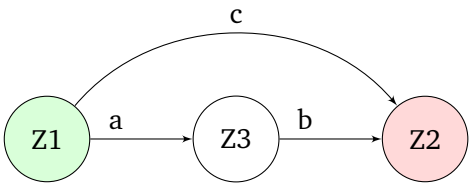


Figura 6.13: Uno de los posibles grafos resultantes de la expansión del grafo de la figura 6.11. Para llegar a esta figura, por un lado, se han evaluado los modificadores, y por otro lado, se han sustituido las superzonas (definidas por los grafos de la figura 6.12). Para realizar estos procesos se obtiene un número aleatorio, que se compara con las probabilidades definidas en los modificadores y en los grafos. Los nodos verdes son las entradas y los rojos las salidas.

Finalmente, la figura 6.13 muestra uno de los posibles resultados del proceso de expansión que ha sufrido el grafo inicial. Para llegar a este punto, primero se han evaluado los modificadores en orden de definición. En este caso, el primero no ha sido superado (50 % de posibilidades) por lo que tanto *S2* como todos sus enlaces han sido eliminados del grafo. Por su parte, el segundo modificador sí ha sido superado (como no podría ser de otra manera, ya que al no existir *S2* la probabilidad es del 100 %) por lo que el enlace *c* se mantiene en el grafo final.

Una vez evaluados todos los modificadores, se ha pasado a expandir las superzonas del grafo (en este caso solamente queda *S1*, ya que *S2* ha sido eliminada). Para ello, se selecciona uno de los posibles grafos de la superzona mediante las probabilidades que tienen asignadas (ver figura 6.12), se evalúan sus modificadores (que en este caso no existen), y se sustituye el nodo en el grafo final. Para ello, se ha seleccionado el primero de los grafos de la superzona, en el cual solo hay un nodo (*Z1*) que es entrada y salida

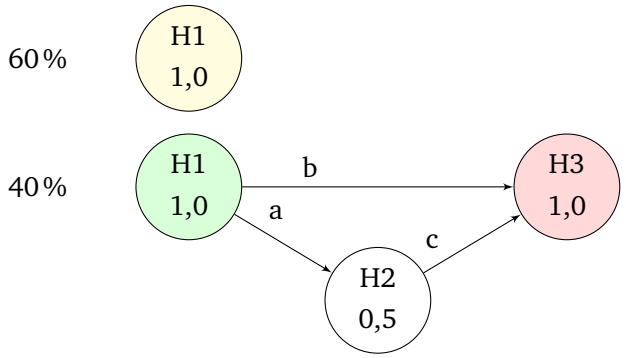


Figura 6.14: Grafos de las zonas de tipo *Z* del grafo de la figura 6.11. Está compuesta por dos subgrafos. Los nodos de tipo *H* son habitaciones, que tienen asignado un peso que representa el tamaño deseado para dicha habitación. Los nodos verdes son las entradas, los rojos las salidas y los amarillos son nodos entradas y salidas a la vez. En este caso, estos grafos no tienen asignados ningún modificador.

a la vez. Para sustituirlo es necesario unir todos los enlaces de entrada a la superzona a los nodos de entrada del grafo a insertar, y todos los enlaces de salida de la superzona a los nodos de salida del grafo a insertar.

Paso 2: Generación de zonas

El siguiente paso en el proceso comprende la construcción de las zonas que van a integrarse en la mazmorra. Para ello, es necesario construir todas aquellas zonas que se encuentran en el grafo expandido obtenido en el paso anterior. Este proceso se explica en los siguientes pasos (2.a, 2.b, 2.c y 2.d), que se repiten para cada zona a construir.

Una vez terminados estos pasos para todas las zonas, ya estarían generados los grafos locales de todas las zonas que van a componer la mazmorra.

2.a Obtención del grafo final de la zona

Para construir cada una de estas zonas es necesario conocer el número y disposición de las habitaciones que la componen. Para ello, se selecciona el grafo que define la zona mediante el mismo proceso que la selección del grafo que define una superzona (selección del grafo mediante su probabilidad y evaluación de sus modificadores).

La figura 6.14 muestra los grafos que definen las zonas de tipo *Z* del grafo que se muestra en la figura 6.13. En este caso cada nodo representa una habitación y tiene un peso asignado que indica el tamaño relativo deseado para dicha habitación.

2.b Posicionamiento de las habitaciones

Una vez obtenido el grafo final de la zona con los modificadores evaluados, se comienza la construcción física de las habitaciones. Como ya hemos dicho anteriormente, este proceso se basa en algoritmos de presión, por lo que el primer paso es posicionar las semillas en un espacio bidimensional dividido por una rejilla regular, desde las cuales van a crecer las habitaciones. Este algoritmo sigue los siguientes pasos:

1. Obtener todas las entradas del grafo y almacenarlas en una colección de habitaciones a posicionar.
2. Obtener una habitación aleatoria de la colección de habitaciones a posicionar, eliminarla de dicha colección, posicionarla en el espacio y añadirla en la colección de habitaciones ya posicionadas. El proceso para posicionar la habitación lo vemos más adelante.
3. Si la colección de habitaciones a posicionar está vacía, continuar con el siguiente punto, en caso contrario, volver al punto 2.
4. Obtener todos aquellos nodos del grafo que tengan un enlace directo con los nodos de las habitaciones ya posicionadas y almacenarlas en la colección de habitaciones a posicionar. Si después de este paso esta colección sigue vacía el proceso se da por terminado, en caso contrario, volver al punto 2.

De esta manera, las habitaciones se posicionan en el espacio de manera ordenada y junto a las habitaciones con las que están relacionadas. Este posicionamiento se realiza en un espacio dividido en pisos, con la restricción de que cada habitación solamente puede estar contenida en un único piso, siendo imposible generar habitaciones que ocupen múltiples pisos de manera vertical.

El procedimiento exacto para posicionarlas depende del número de dependencias[1] que tenga la habitación que estamos posicionando con las habitaciones ya posicionadas.

I. La habitación no tiene dependencias

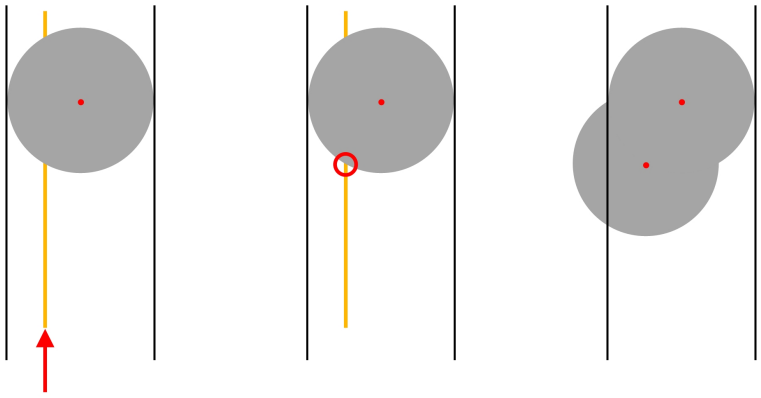
En caso de que la habitación a posicionar no tenga dependencias con las habitaciones ya posicionadas (únicamente posible en el caso de algunas de las entradas al grafo),

[1] Utilizamos el término dependencia para hacer referencia a un enlace entre los nodos de dos zonas o habitaciones de un mismo grafo, independientemente de la dirección de dicho enlace. Así, cuando decimos que una habitación es dependiente de otra, nos referimos a que existe un enlace entre las dos.

existen dos posibilidades: que la habitación sea la primera habitación de la zona o que ya existan otras entradas posicionadas. En los dos casos, además de la propia semilla, se establece un área de seguridad a su alrededor, en la que no va a posicionarse ninguna otra semilla y cuyo radio viene determinado por la fórmula $\sqrt{TH}$, donde TH es el tamaño global de las habitaciones establecido en los parámetros de entrada (podemos ver estas áreas de seguridad en las imágenes de la figura 6.15 pintadas en gris alrededor de cada semilla posicionada).

En caso de que sea la primera habitación de la zona en posicionarse, la habitación se sitúa en la celda $[0,0]$, debido a que esta es la habitación que marca el sistema de referencia para el resto de habitaciones.

En caso de que ya existan otras habitaciones posicionadas, los pasos a seguir son los siguientes (ver figura 6.15):



(a) Selección de la columna aleatoria. (b) Obtención de la altura. (c) Resultado.

Figura 6.15: Proceso de posicionamiento de habitaciones sin dependencia con habitaciones ya posicionadas.

1. Inicialmente se delimita el espacio de posicionamiento mediante dos líneas imaginarias verticales situadas en los límites del área de seguridad de la primera habitación en el espacio. Podemos ver estas líneas en la subfigura 6.15(a) coloreadas en negro. El punto rojo en esta imagen es la primera habitación posicionada para esta zona.

2. A continuación, se obtiene una columna aleatoria de la rejilla entre las dos líneas obtenidas en el paso anterior. En la subfigura 6.15(a) podemos ver esta línea

coloreada en naranja y señalada mediante un flecha roja.

3. Después se obtiene el punto más alto de esa columna con las siguientes condiciones:

   - No debe encontrarse dentro de una zona de seguridad de otra habitación.
   - Ninguno de los puntos de la columna seleccionada por debajo del punto seleccionado se debe encontrar dentro de un área de seguridad de otra habitación.

   Para cumplirlas, se recorre la columna desde abajo hacia arriba, obteniendo el último punto de dicha columna antes de entrar en el área de seguridad de otra habitación. La subfigura 6.15(b) muestra este punto marcado por un círculo rojo.

4. Finalmente, se sitúa la semilla de la nueva habitación en dicho punto y se crea su área de seguridad, como podemos ver en la subfigura 6.15(c).

II. La habitación tiene dependencias

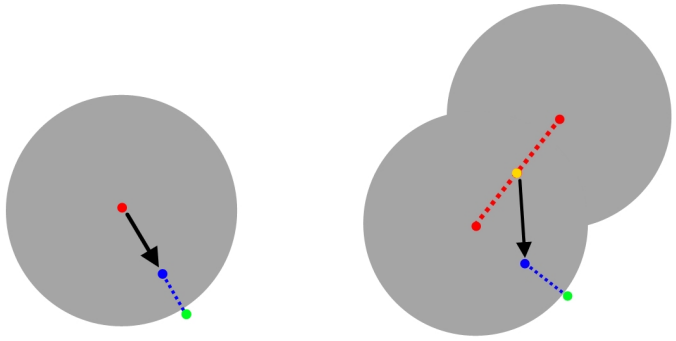
Si la habitación a posicionar tiene al menos un enlace con las habitaciones ya posicionadas, el procedimiento comienza obteniendo el punto medio entre todas aquellas habitaciones ya situadas con las que tiene relación la nueva habitación (en la subfigura 6.16(a) tiene una sola dependencia, por lo que el punto medio es la posición de esta única habitación; en la subfigura 6.16(b) se muestra coloreado en amarillo el punto medio entre dos habitaciones para posicionar una habitación con dos dependencias).

En caso de que las dependencias de una habitación se encuentren distribuidas en diferentes pisos, la nueva habitación se situará en el piso donde haya un mayor número de estas. Además, para obtener el punto medio de dichas dependencias, también se tienen en cuenta las posiciones de las habitaciones que se encuentran en otros pisos como si estuvieran situadas en el mismo piso.

A continuación, se aplica un desplazamiento aleatorio sobre este punto (representado mediante una flecha negra en la figura 6.16), cuya longitud máxima es el propio radio del área de seguridad.

Finalmente, el proceso termina obteniendo el punto libre[1] más cercano al punto obtenido en el paso anterior (coloreado en verde en figura 6.16), posicionando allí la nueva habitación y estableciendo su nueva área de seguridad.

[1]Consideramos punto libre a cualquier punto que no se encuentre dentro del área de seguridad de otra habitación



(a) Posicionamiento de semilla con una dependencia.

(b) Posicionamiento de semilla con varias dependencias.

Figura 6.16: Proceso de posicionamiento de semillas dependientes de semillas ya posicionadas. El punto verde es el punto donde va a posicionarse la nueva semilla, mientras que los rojos son las semillas con las cuales mantiene una relación la nueva semilla a posicionar.

Existe una excepción a este proceso que se produce cuando el punto obtenido mediante el desplazamiento aleatorio se encuentra más cerca de un número N de habitaciones ya posicionadas no relacionadas con la nueva habitación a posicionar que a un punto libre en el espacio. N es el número de habitaciones ya posicionadas en el espacio relacionadas con la habitación a posicionar. Podemos ver un ejemplo de esta situación en la figura 6.17.

En estos casos, el método cambia de piso (al superior o al inferior) pero mantiene el punto desde el que se busca el punto libre más cercano (de color verde en la figura 6.17). Este proceso normalmente soluciona el problema, aunque si vuelve a darse la misma situación, se vuelve a cambiar de piso, hasta que finalmente se pueda posicionar la habitación.

2.c Crecimiento de las habitaciones

El objetivo de este paso es obtener la forma final de las habitaciones de la zona y, para ello, utilizamos los algoritmos de presión que hemos visto previamente.

En este punto la zona está compuesta por una serie de planos bidimensionales (una por cada piso) divididos por una rejilla regular, sobre los cuales se encuentran distribuidas las semillas de las diferentes habitaciones que la componen.

El proceso se realiza para cada piso de manera independiente, y comienza mediante

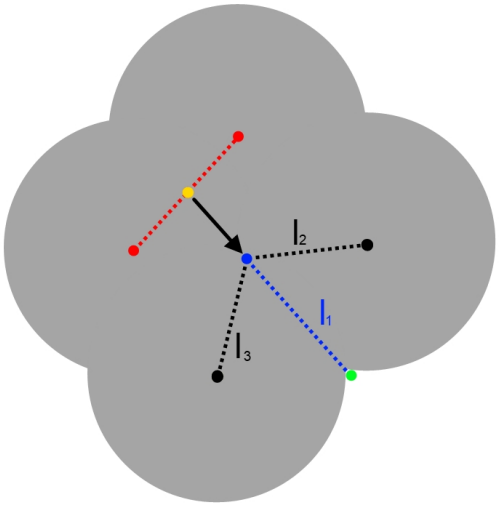


Figura 6.17: Situación en la cual no puede posicionarse la habitación, ya que el número de dependencias (N) es igual a 2, y el punto obtenido se encuentra más cerca de 2 habitaciones no dependientes que de un punto libre en el espacio ($\ell 1$ es mayor que $\ell 2$ y $\ell 3$). Las habitaciones dependientes con la nueva habitación se encuentran coloradas en rojo, mientras que las no dependientes se han coloreado en negro. El punto azul es el punto desde el cual se busca el punto libre más cercano.

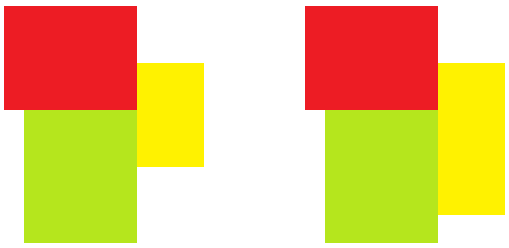
el crecimiento rectangular de las habitaciones, en el que van aumentando de tamaño poco a poco de manera iterativa y todas a la vez, hasta alcanzar el área deseada (obtenida multiplicando el tamaño relativo de la habitación por el tamaño global de las habitaciones) o encontrarse con otra habitación (caso en que las dos habitaciones dejan de crecer en la dirección y sentido en que se encuentra la otra habitación, pero continúan creciendo en el resto de sentidos).

Este crecimiento se realiza con una relación de aspecto entre el ancho y el alto de cada habitación determinado, siempre que no se encuentren otras habitaciones en el camino, caso en el que se podría ver modificado (por ejemplo, si una habitación encuentra dos habitaciones en sentidos opuestos, ya que deja de crecer en los dos sentidos de esa dirección). Esta relación de aspecto se obtiene de manera aleatoria y su máximo y mínimo pueden configurarse libremente (en esta investigación la hemos limitado para que ninguna habitación sea más del doble de alta que de ancha o viceversa).

A continuación, se realiza un proceso de crecimiento en L en todas aquellas habitaciones que no hayan alcanzado su área deseada en el proceso de crecimiento anterior. Además, utilizando el parámetro de entrada de probabilidad de crecimiento en L, se comprueba si alguna habitación que ya haya alcanzado su tamaño deseado va a sufrir

también este crecimiento.

Antes de que una habitación crezca en L es necesario comprobar que uno de sus lados pueda crecer de esta manera, es decir, que una de sus cuatro paredes esté parcialmente ocupada por otra habitación. Es posible configurar la cantidad de espacio libre que una pared debe tener para poder crecer en L, aunque para mantener una forma realista, en esta investigación la hemos establecido de forma empírica para que sea del 30 % (es decir, para que un lado de una habitación pueda sufrir un crecimiento en L debe tener, al menos, el 30 % de su longitud libre al otro lado de la pared). La figura 6.18 muestra una comparación un caso en el que la habitación puede crecer en L y otro caso en el que no.



(a) (b)

Figura 6.18: Restricciones para el crecimiento en L. En la subfigura (a) la habitación verde puede crecer en L, ya que la habitación amarilla le deja libre más del 30 % de su pared. Por el contrario, en la subfigura (b) esto no es así, por lo que no puede crecer en L en esa dirección.

Una vez determinadas las habitaciones que van a crecer en L y la dirección en que van a hacerlo, se procede a realizar dicho crecimiento habitación por habitación en un orden aleatorio. La distancia de crecimiento se establece mediante los siguientes factores:

- Si la habitación que se va a ampliar no ha llegado todavía su área deseada, crecerá lo necesario para alcanzarla.

- Si la habitación ha alcanzado su área deseada y ha superado la probabilidad de crecimiento en L establecida por el usuario, entonces, crecerá hasta alcanzar un área aleatoria hasta un máximo configurable por el diseñador (en nuestro caso lo hemos establecido de forma empírica como el 30 % del área total de la habitación).

- Si en cualquiera de los dos casos anteriores el crecimiento en L no puede realizarse (debido a que ha encontrado otra habitación en el camino), crecerá el máximo posible.

(a) Semillas posicionadas. (b) Crecimiento rectangular por presión. (c) Crecimiento en L.

Figura 6.19: Proceso de crecimiento de las habitaciones de la zona *Z* definida por el grafo de tres habitaciones que se muestra en la figura 6.14.

La figura 6.19 muestra las fases de crecimiento de las habitaciones: en la subfigura 6.19(a) el proceso comienza con las semillas posicionadas en el espacio, en la subfigura 6.19(b) continua realizando el crecimiento rectangular, y en subfigura 6.19(c) finaliza con el crecimiento en L.

Una vez terminado este proceso, se sitúan todas las paredes entre las habitaciones.

2.d Generación de pasillos y puertas

La generación de cada zona independiente se finaliza asegurando la conectividad entre las habitaciones enlazadas del grafo. Este proceso se realiza en dos pasos:

Primero, se buscan todas aquellas parejas de habitaciones que deben estar conectadas y están contiguas pared con pared, y se construye una puerta en un punto aleatorio de la pared que las conecta.

A continuación, se buscan aquellas parejas de habitaciones que deben estar conectadas pero que no están contiguas pared con pared. La figura 6.20 muestra la zona de la figura 6.19 con las paredes y las puertas ya posicionadas.

Por cada pareja de habitaciones a conectar que no estén contiguas pared con pared hay dos posibilidades: que se encuentren en el mismo piso o que no.

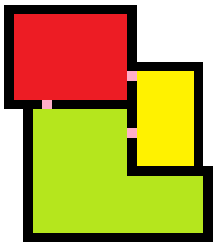


Figura 6.20: Zona completamente conectada mediante de puertas.

I. Las habitaciones están en el mismo piso

En este caso se construye un pasillo entre ellas utilizando el algoritmo A* (ver sección 6.3.2.2) con las siguientes particularidades:

- Las únicas casillas válidas que puede cruzar el camino son: casillas de la habitación inicial o final, paredes que no estén junto a una puerta o que no pertenezcan ya a otro pasillo (para evitar que la construcción del nuevo pasillo pueda cegar otra salida) y casillas libres.
- Solamente están permitidos los desplazamientos en horizontal o en vertical.
- Se establece como punto inicial y final un punto aleatorio de cada habitación.
- El coste de atravesar una casilla de cualquiera de las dos habitaciones es 0, mientras que el coste de atravesar cualquier otro tipo de casilla valida es 1. De esta manera, el camino siempre va a ser el más corto posible entre las dos habitaciones.

En este punto hay dos posibilidades: que el algoritmo A* encuentre un camino (caso en el que se construye y se pasa a la siguiente pareja de habitaciones a conectar), o que no lo encuentre. Caso en el que se actúa de la siguiente manera:

1. Se selecciona el piso inmediatamente inferior o superior al piso de las habitaciones.
2. Se buscan dos puntos, cada uno situado en una de las habitaciones, cuyo valor en el piso seleccionado en el punto anterior sea válido para el algoritmo A*.
3. Se busca un camino entre esos dos puntos en el piso seleccionado anteriormente utilizando el algoritmo A*. Si no es posible encontrarlo, se cambia de piso otra vez y se vuelve a buscar. Si es imposible encontrar un camino tanto en el piso inferior como en el superior, se reinicia la generación de la zona.

4. Una vez encontrado el camino en el piso contiguo, se construye el pasillo con escaleras en su inicio y final que lo conectan con las habitaciones.

Así, las habitaciones quedan conectadas por un pasillo superior o inferior.

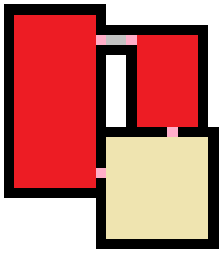
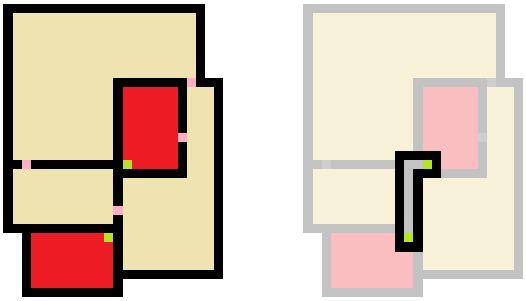


Figura 6.21: Pasillo simple entre dos habitaciones (en rojo) del mismo piso.

La figura 6.21 muestra dos habitaciones conectadas por un pasillo simple creado en el mismo piso.



(a) Piso de las habitaciones.

(b) Piso superior en el que se ha creado el pasillo.

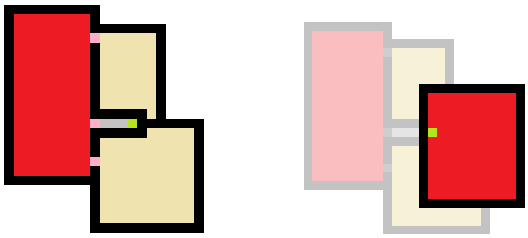
Figura 6.22: Pasillo entre dos habitaciones (en rojo) imposibles de conectar sin cambiar de piso. Las escaleras se encuentran coloreadas en verde.

En la figura 6.22 podemos ver la conexión entre dos habitaciones del mismo piso cuando es imposible crear un camino directo. Podemos observar como en la subfigura 6.22(a) es imposible crear un pasillo con las características expuestas previamente para el algoritmo de búsqueda de caminos debido a la disposición de las otras dos habitaciones y de las puertas que las conectan. La subfigura 6.22(b) muestra el piso superior con el pasillo creado.

II. Las habitaciones están en pisos diferentes

Cuando hay que crear una conexión entre dos habitaciones en diferentes pisos existen varias posibilidades:

- Existe al menos un punto compartido por ambas habitaciones (aunque cada una en su piso), entre los cuales todos los pisos intermedios tienen un valor válido para el algoritmo A*. En este caso se crea una escalera directa entre las dos habitaciones, atravesando los pisos necesarios.
- No existe un punto que cumple el caso anterior, por lo que se busca un punto en una de estas habitaciones en el que se pueda ir al piso de la otra habitación mediante una escalera. Caso en que se crea una escalera en ese punto entre una de las habitaciones y el piso de la otra, y se conecta después dicho punto con la otra habitación mediante un pasillo (con el algoritmo A* previamente descrito).
- No existe un punto en ninguna de las habitaciones que cumple ni el primer ni el segundo caso. Así, el proceso de construcción de la zona se reinicia.



(a) Piso con una de las habitaciones donde se ha creado el pasillo.

(b) Piso superior en el que está la otra habitación.

Figura 6.23: Pasillo entre dos habitaciones (en rojo) situadas en diferentes pisos. Las escaleras se encuentran coloreadas en verde.

La figura 6.23 muestra dos habitaciones situadas en pisos diferentes y conectadas por un pasillo, en el que en uno de sus extremos hay una puerta a una de las habitaciones, mientras que en la otra hay una escalera a la otra habitación. Esta es la solución cuando se da el segundo caso. Si se diese el primero, el pasillo no existiría, y se unirían las habitaciones mediante una escalera directa.

Este procedimiento es el último paso de la generación de cada zona. Una vez se han generado todas las zonas, la construcción de la mazmorra continúa con el paso 3.

Paso 3: Generación de la estructura de la mazmorra

En este paso se crea el esqueleto de la mazmorra, definiendo la posición de las zonas y generando los pasillos principales que las conectan.

En primer lugar se establece la posición de las zonas mediante el mismo algoritmo que para posicionar las semillas de las habitaciones, aunque con las siguientes particularidades:

- El algoritmo se limita a situar las zonas en un solo piso, obviando el motivo por el cual se podía situar una habitación en otro piso.
- El área de seguridad de cada zona viene determinada por su tamaño real (obtenido al generar físicamente cada una de las zonas) y el parámetro global que determina la distancia entre zonas.

Una vez establecida la posición de cada zona se crean los pasillos necesarios para conectar aquellas zonas enlazadas en el grafo expandido. Este proceso se realiza mediante el algoritmo A* modificado (ver sección 6.3.2.2) con las siguientes particularidades:

- El pasillo puede cruzar cualquier casilla, excepto las que pertenezcan a una zona diferente a la inicial y final.
- Solamente están permitidos los desplazamientos en horizontal o en vertical.
- Se establece un punto aleatorio de cada zona como punto inicial y final.
- El coste de atravesar una casilla de cualquiera de las dos zonas es 0, mientras que el coste de atravesar cualquier otro tipo de casilla válida es 1. De esta manera, el camino siempre va a ser el más corto posible entre las dos zonas.

La figura 6.24 muestra el resultado de este proceso.

Una vez generado cada camino, se construye el pasillo, que si se cruza con otro pasillo ya construido, se sube o se baja un piso mediante unas escaleras creando una galería inferior o superior hasta que sea posible retornar al piso original.

Paso 4: Inserción de zonas

A continuación se insertan las zonas construidas previamente en sus respectivos huecos, posicionando el piso 0 de cada una de ellas en el piso donde se han construido los

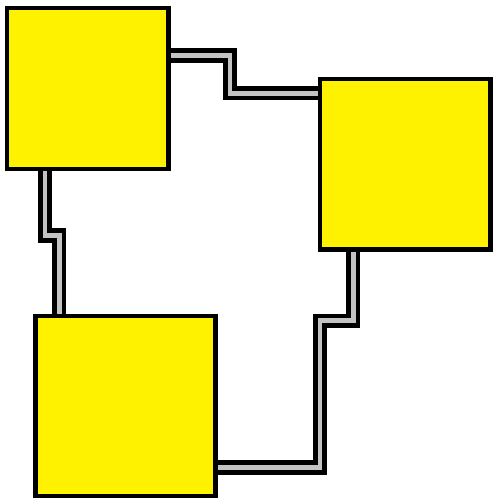


Figura 6.24: Esqueleto de la mazmorra. Las zonas están coloreadas en amarillo. Los pasillos que las unen son los pasillos principales de la mazmorra.

pasillos entre zonas en el paso anterior; y la creación de los pasillos necesarios desde las entradas hasta las habitaciones de entrada, y desde las habitaciones de salida hasta las salidas de cada zona.

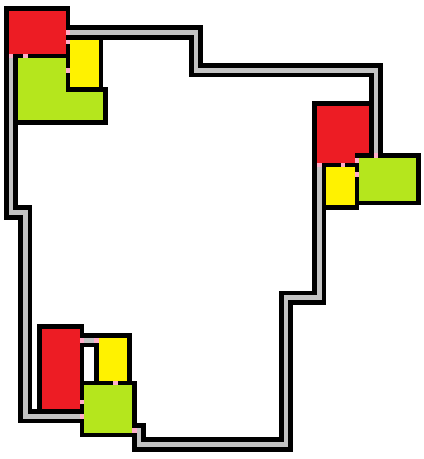


Figura 6.25: Resultado final. Esta mazmorra ha sido generada utilizando los grafos de ejemplo mostrados en el primer paso de esta sección.

Paso 5: Creación de la capa jugable de la mazmorra

Una vez construida la mazmorra al completo con todas las habitaciones definidas e identificadas, se asignan los eventos jugables definidos por el diseñador a cada una de ellas. La figura 6.26 muestra una mazmorra con los eventos jugables asignados.

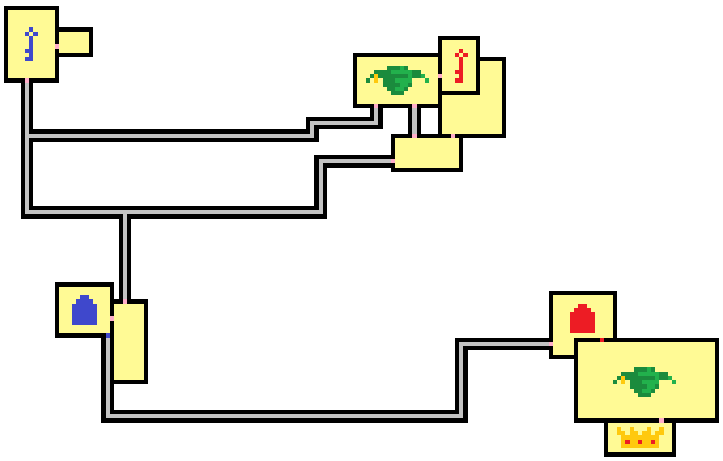


Figura 6.26: Mazmorra final con los eventos jugables de cada habitación asignados. Podemos ver la situación de las llaves y las puertas que abren (organizadas por colores), de los enemigos, y del tesoro final.

6.3.5 Resultados

En esta sección se muestran algunos ejemplos de mazmorras generadas con el método propuesto, además de los resultados obtenidos en una serie de experimentos que evalúan su rendimiento. Abordamos estos resultados desde un punto de vista cualitativo en el capítulo 7.

Debido al alto número de variables que determinan las características de una mazmorra, la evaluación de la velocidad de generación se ha divido en varios apartados, estableciendo diferentes conjuntos de configuraciones para el algoritmo en cada uno de ellos.

Para realizar la medición, se ha ejecutado el algoritmo 200 veces con cada configuración y se ha calculado después la media, la desviación típica y la desviación media de cada conjunto de tiempos[1].

[1]El método ha sido ejecutado en una CPU Intel©Core™ i7 950, con una frecuencia de reloj de 3,07GHz.

6.3.5.1 **Rendimiento respecto al número de zonas**

Para medir el rendimiento del algoritmo respecto al número de zonas que componen una mazmorra, se han realizado mediciones con mazmorras de 1, 2, 3 y 4 zonas conectadas de manera encadenada. Respecto al resto de parámetros, el tamaño mínimo de las habitaciones es de 100 casillas y la distancia entre las zonas es como mínimo de 50 casillas. Todas las zonas están compuestas de una sola habitación.

Número de zonas	Tiempo medio (ms)	Desviación estándar	Desviación media
1	119,67	1,52	0,91
2	1744,94	1999,87	1471,64
3	3237,82	2912,04	2238,25
4	5536,50	5150,86	3168,47

Tabla 6.9: Rendimiento respecto al número de zonas.

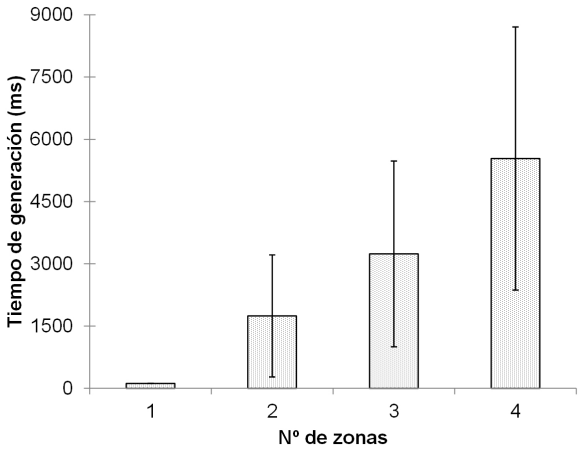


Figura 6.27: Rendimiento respecto al número de zonas.

La tabla 6.9 y la figura 6.27 muestran los resultados de estos experimentos.

Como podemos ver, el aumento del número de zonas causa una enorme variación en el rendimiento y tiene un fuerte impacto en el rendimiento global del algoritmo. Por otro lado, cuando solo se construye una zona, la variación es prácticamente nula, debido a que el factor que introduce dicha variación no es la construcción de las zonas en sí, sino la ejecución del algoritmo de búsqueda de caminos utilizado para construir los pasillos entre ellas.

6.3.5.2 Rendimiento respecto al número de habitaciones

Para medir el rendimiento del algoritmo respecto al número de habitaciones que componen cada zona, se han realizado mediciones con mazmorras de una sola zona compuesta por 1, 2, 3 y 4 habitaciones conectadas completamente entre ellas. Respecto al resto de parámetros, el tamaño mínimo de las habitaciones es de 100 casillas.

Número de habitaciones	Tiempo medio (ms)	Desviación estándar	Desviación media
1	119,67	1,52	0,91
2	123,38	1,79	1,07
3	137,57	2,09	1,74
4	163,18	13,32	8,61

Tabla 6.10: Rendimiento respecto al número de habitaciones.

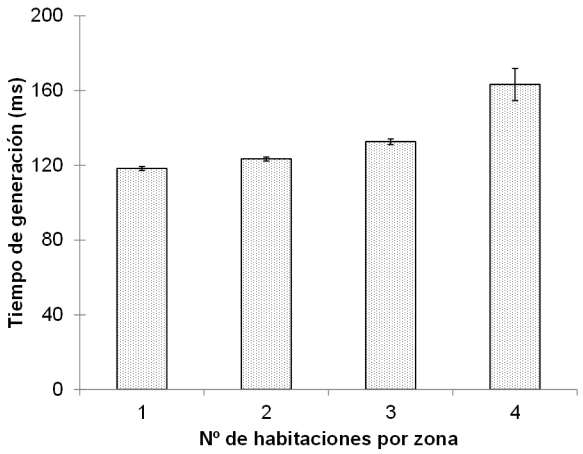


Figura 6.28: Rendimiento respecto al número de habitaciones.

La tabla 6.10 y la figura 6.28 muestran los resultados de estos experimentos.

Podemos observar como al aumentar el número de habitaciones en cada zona, el tiempo necesario aumenta gradualmente al igual que su variabilidad. Este aumento incremental de la variabilidad viene dado principalmente por el incremento del número de conexiones entre las habitaciones (ya que en el experimento todas las habitaciones se encuentran conectadas con el resto).

6.3.5.3 **Rendimiento respecto al tamaño de las habitaciones**

Para medir el rendimiento del algoritmo respecto al tamaño de las habitaciones que componen cada zona, se han realizado mediciones con mazmorras cuyas habitaciones tienen un tamaño mínimo de 25, 50, 75 y 100 casillas. Todas las mazmorras generadas están compuestas de una sola zona que a su vez se compone de tres habitaciones completamente conectadas entre ellas.

Tamaño de las habitaciones (casillas)	Tiempo medio (ms)	Desviación estándar	Desviación media
25	135,14	2,15	1,72
50	135,46	2,13	1,81
75	136,32	2,16	1,82
100	137,57	2,09	1,74

Tabla 6.11: Rendimiento respecto al tamaño de las habitaciones.

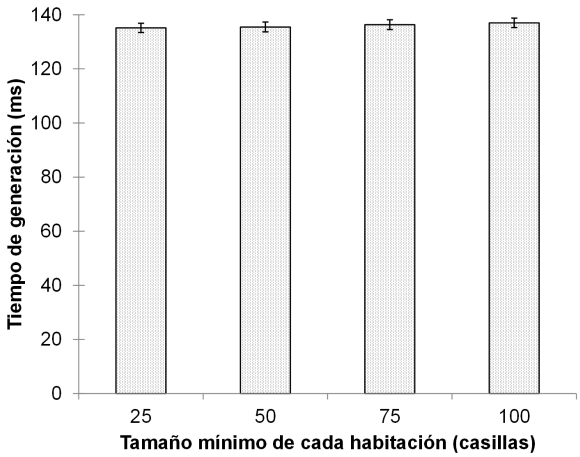


Figura 6.29: Rendimiento respecto al tamaño de las habitaciones.

La tabla 6.11 y la figura 6.29 muestran los resultados de estos experimentos, en los que podemos comprobar que aunque el algoritmo tarda más en ejecutarse si se incrementa el tamaño de las habitaciones, este factor influye levemente en rendimiento global.

6.3.5.4 **Rendimiento respecto a la distancia entre zonas**

Para medir el rendimiento del algoritmo respecto a la distancia entre las diferentes zonas que componen una mazmorra, se han realizado mediciones con mazmorras formadas únicamente por dos zonas separas por una distancia mínima de 25, 50, 75 y 100 casillas. Todas las zonas están formadas por una única habitación de un tamaño mínimo de 100 casillas.

Distancia entre zonas (casillas)	Tiempo medio (ms)	Desviación estándar	Desviación media
25	835,47	908,03	638,82
50	1744,94	1999,87	1471,64
75	4840,15	5371,92	4421,37
100	9870,46	12961,22	9940,10

Tabla 6.12: Rendimiento respecto a la distancia entre zonas.

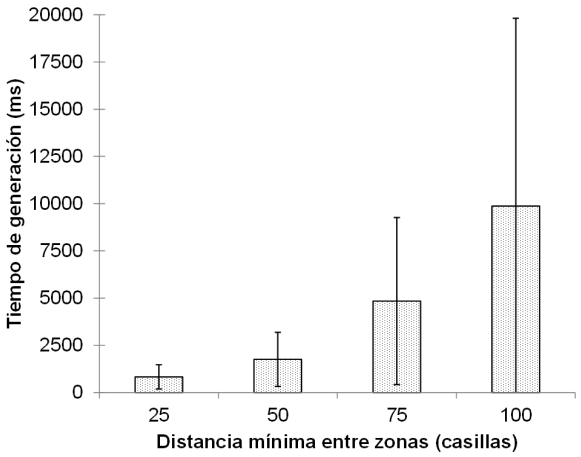


Figura 6.30: Rendimiento respecto a la distancia entre zonas.

La tabla 6.12 y la figura 6.30 muestran los resultados de estos experimentos.

Podemos comprobar como la distancia entre las zonas es un factor crucial para el rendimiento global del método, ya que el algoritmo de búsqueda de caminos se ve altamente afectado por los aumentos de tamaño del espacio en el que se ejecuta.

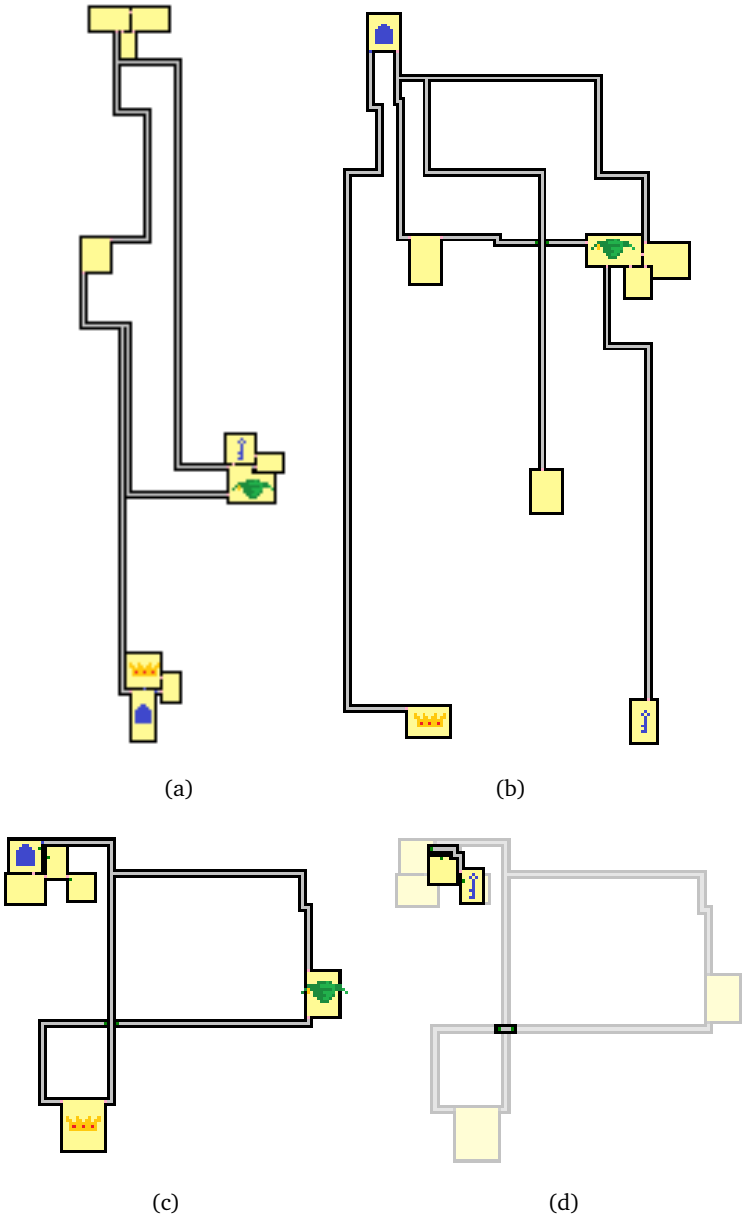


Figura 6.31: Ejemplos de mazmorras generadas con nuestro método. (a) y (b) son mazmorras independientes. (c) y (d) son pisos de la misma mazmorra. La mazmorra (b) tiene un piso adicional, omitido debido a que solo contiene la parte superior de los pasillos que se cruzan.

6.3.5.5 **Ejemplos**

Las figuras 6.26 y 6.31 muestran algunas mazmorras de ejemplo obtenidas con nuestro método.

Podemos observar la estructura que presentan las mazmorras obtenidas con nuestro método, las cuales se dividen en habitaciones conectadas mediante puertas y pasillos que a su vez se agrupan formando diferentes zonas conectadas mediante pasillos principales. Además, las figuras 6.31(c) y 6.31(d) muestran una mazmorra formada por dos pisos en la que una de las zonas tiene sus habitaciones distribuidas en plantas diferentes.

Asimismo, estas figuras muestran cómo se estructura la jugabilidad de la mazmorra en capas de objetivos. Por ejemplo, la mazmorra que se muestra en la figura 6.26 tiene como objetivo global alcanzar la sala del tesoro (representada mediante una corona en la zona inferior derecha). Además, cada zona tiene su propio objetivo particular, como obtener una determinada llave (en las dos zonas superiores). Finalmente, podemos observar que algunas las habitaciones también tienen su propio objetivo, como superar a un enemigo (representado en la figura mediante la cara de un monstruo). De la misma manera, las mazmorras de la figura 6.31 presentan unas estructuras jugables similares.

«Pastel y apoyo psicológico estarán disponibles al final de la prueba.»

GLaDOS (*Portal*, Valve, 2007)

CAPÍTULO

7

Análisis de resultados y conclusiones

En esta investigación proponemos un método para la generación procedural de mundos virtuales formados por terrenos volumétricos (capítulos 4, 5 y 6). Además, analizamos el estado actual del área de la generación procedural tanto en el mundo científico como en el industrial, y extraemos una serie de requisitos que todos los generadores procedurales deberían cumplir y unas métricas para facilitar su evaluación.

En este capítulo analizamos los resultados obtenidos por los métodos propuestos y extraemos una conclusión de toda la investigación. Se estructura de la siguiente manera: en la sección 7.1 evaluamos el éxito del método propuesto utilizando para ello el conjunto de métricas y requisitos definidos en el capítulo 3, y en la sección 7.2 lo comparamos con otros generadores similares; en la sección 7.3 enumeramos las fortalezas y debilidades de nuestro método; en la 7.4 comparamos el resultado final de la investigación con la hipótesis inicial; en la sección 7.5 enumeramos las principales contribuciones de esta investigación; en la sección 7.6 proponemos unas posibles líneas en las que trabajar para continuar con esta investigación; y en la 7.7 exponemos un conjunto de consideraciones finales extraídas del conocimiento adquirido durante el desarrollo de esta investigación.

7.1 Comparación con los requisitos iniciales

En esta sección analizamos los métodos propuestos en esta investigación partiendo de los requisitos presentados en el capítulo 3, que comparamos con las características de los métodos que proponemos utilizando las métricas establecidas previamente.

Para poder afirmar definitivamente que el método supera estos requisitos son necesarios los siguientes experimentos: por un lado, para evaluar los aspectos orientados al diseñador, es necesario realizar una valoración mediante una colección de experimentos en los que un conjunto de diseñadores utilicen el método que proponemos para generar mundos virtuales; por otro, para evaluar los aspectos orientados al jugador, es necesario implementar el método en una serie de aplicaciones interactivas (un videojuego, por ejemplo) y realizar una valoración mediante una colección de experimentos en los que un conjunto de jugadores lo utilicen.

Debido a que ambos conjuntos de experimentos quedan fuera del alcance de esta tesis, realizamos una primera aproximación mediante el análisis cualitativo del método, dejando dichos experimentos para una futura ampliación de esta investigación.

Realizamos este análisis sobre los métodos descritos en los capítulos 4, 5 y 6, por lo que en los siguientes apartados se hacen múltiples referencias a los resultados y secciones de dichos capítulos.

Innovación

> **Requisito: 2**. Los resultados obtenidos presentan elementos inesperados, además de diferencias significativas entre elementos del mismo tipo.

Tanto el método de generación de terrenos como el generador de cuevas propuestos en esta tesis, se basan en probabilidades para definir qué elemento hay que generar en cada momento y para determinar su configuración.

En primer lugar, el generador de terrenos básicos presenta las siguientes características que favorecen la innovación:

- Como vemos en la figura 5.12, los terrenos resultantes están formados tanto por capas como por vetas de mineral. La decisión de qué elemento construir en cada momento (capa de materiales o veta de minerales) se toma mediante la evaluación de una probabilidad basada en la densidad de vetas en el terreno definida

por el diseñador (ver sección 5.4.2), por lo que es imposible predecir cuántas vetas y cuántas capas van a crearse, ni cómo están ordenadas entre sí.

En los experimentos llevados a cabo hemos establecido una densidad de vetas de 0,5 (ver tabla 5.2), y los resultados muestran que se crean, de media, alrededor de 4,8 capas y 1,7 vetas por cada terreno. Podemos comprobar que tanto el número de capas como de vetas varían entre los diferentes terrenos, ya que los resultados muestran cifras decimales.

- Las capas y vetas están formadas por una colección de materiales con diferentes concentraciones cada uno. Como vemos en la sección 5.4.2, la colección de materiales se determina mediante un método probabilístico definido por las afinidades de los materiales cercanos. Podemos comprobar esto en la figura 5.12, que muestra terrenos cuyas capas combinan diferentes materiales. Por ejemplo, la capa amarilla del terreno de la subfigura 5.12(b) está compuesta por la mezcla de otros dos materiales, ya que en los datos de entrada utilizados para generar este terreno no se incluye un material de color amarillo.

  Este hecho hace que, a no ser que los datos de entrada sean muy restrictivos, sea imposible predecir la configuración de materiales de las capas y vetas que va a tener el terreno, por tanto, es común que se creen elementos con características no definidas en los valores de entrada.

- Como vemos en la sección 5.4.2, la forma física de cada capa de mineral se define, además de por la forma de la capa anterior, por un generador de ruido Perlin, cuyos resultados son imposibles de predecir.

- La forma de cada veta mineral se genera mediante un algoritmo fractal con un grado muy alto de aleatoriedad (proceso explicado en la sección 5.4.2.2). La subfigura 5.9(c) muestra un ejemplo real de la forma obtenida con este método, en la que podemos comprobar el tipo de formas que se consiguen.

En segundo lugar, la innovación en el generador de sistemas de cuevas viene dada por las siguientes características:

- Como hemos visto, los sistemas de cuevas se construyen mediante la generación y expansión de un grafo cuyos nodos son diferentes puntos interesantes para la jugabilidad (POIs). La decisión de qué tipo de POI crear en cada momento se toma mediante un método probabilístico basado en las afinidades e incompatibilidades entre los diferentes tipos de POI definidos por el diseñador (ver sección 6.2.4.2).

Al igual que para los materiales de las capas del generador de terrenos, a no ser que los datos de entrada sean muy restrictivos, es imposible predecir cómo van a organizarse en el resultado final.

- Como vemos en la sección 6.2.4.3, las características de dichos POIs (profundidad a la que se encuentran, tamaño de la cámara subterránea, distancia con el POI anterior, número de galerías que unen cada pareja de POIs, etc.) se obtienen aleatoriamente utilizando los rangos definidos por el diseñador.

- La sección 6.2.4.4 muestra que el proceso de generación de las galerías subterráneas que unen cada pareja de POIs incluye cierto grado de sinuosidad (obtenida aleatoriamente dentro de un rango definido por el diseñador). Esta sinuosidad otorga aleatoriedad al proceso de *pathfinding*, por lo que hace imposible predecir el camino que va a generar. La subfigura 6.9(b) muestra un camino con un alto grado de sinuosidad.

- Como vemos en la sección 6.2.4.3 la forma de cada cámara viene dada por la forma de la celda del diagrama de Voronoi en la que se encuentra, mientras que la forma de dicha celda está definida por la distribución de las semillas sobre las cuales se construye el diagrama. Debido a que dichas semillas se distribuyen de una manera aleatoria, la forma de las celdas obtenidas es completamente impredecible. En la figura 6.3 podemos ver un ejemplo del grado de aleatoriedad que manifiesta la forma de las cámaras subterráneas debido a esto.

- Como hemos visto en el primer punto, las cuevas se forman expandiendo un grafo de POIs en función de sus afinidades. Debido a ello, es imposible saber de antemano la forma que va a tomar una cueva, ya que es posible incluso que se extienda infinitamente.

En tercer lugar, nuestro método de generación de mazmorras ve limitada su capacidad de innovación debido a que los elementos que genera presentan unos requisitos mucho más restrictivos. A pesar de ello, los siguientes aspectos ayudan a que la innovación de este generador sea alta:

- Como hemos visto anteriormente, al expandir el grafo principal de una mazmorra se tienen en cuenta una serie de modificadores que determinan la existencia o no de algunos elementos de dicho grafo. Esto potencia la innovación del método, aunque debido al alto grado de control que requiere la creación de las mazmorras es el propio diseñador el que define el grado de incertidumbre y de innovación que tiene al generador.

Podemos ver un ejemplo de esto en el grafo de la figura 6.11, que al expandirse se evalúan sus modificadores y probabilidades, resultando en un grafo impredecible (por ejemplo, el que se muestra en la figura 6.13).

- El posicionamiento de las habitaciones y de las zonas se basa en sus interdependencias, aunque como vemos en la sección 6.3.4 tiene un componente aleatorio muy marcado. La figura 6.25 muestra una mazmorra formada por tres zonas que, aunque son del mismo tipo, tienen sus habitaciones posicionadas de una manera completamente diferente.

- De la misma manera que el algoritmo de posicionamiento, el método de generación de habitaciones mediante presión tiene un fuerte componente azaroso, ya que el resultado depende de la relación de aspecto de cada habitación, que se obtiene aleatoriamente, y de su posición, que como hemos visto en el punto anterior, también se obtiene aleatoriamente. La mazmorra que se muestra en la figura 6.25 es un ejemplo de este hecho, ya que aunque las tres zonas que la componen son del mismo tipo, sus habitaciones presentan diferencias significativas.

Además, este grado de innovación permite generar familias de mazmorras que comparten jugabilidad, pero que tienen diferencias estructurales para que el jugador no sienta más similitudes.

Para finalizar, como hemos visto en el capítulo 4, el método propuesto en esta investigación para representar los terrenos volumétricos no limita de manera alguna el tipo ni las características de las estructuras que un terreno puede contener, además de que facilita la mezcla de materiales en el terreno (ver sección 4.2.3.1).

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de innovación, ya que los mundos virtuales obtenidos por el método propuesto presentan elementos inesperados con diferencias significativas entre ellos.

Usabilidad

Requisito: 1. Los parámetros de entrada están relacionados exclusivamente con el resultado, evitando todos los aspectos técnicos.

En primer lugar, los datos de entrada del generador de terrenos básicos están formados por la siguiente información (podemos ver la descripción detallada de todos ellos en la sección 5.3):

- Descriptores de los materiales: definen los materiales presentes en el terreno.
- Modificadores de los materiales: definen la influencia de cada material en las capas y/o vetas que los contienen.
- Relaciones de los materiales: determinan las afinidades entre materiales.
- Parámetros globales: determinan aspectos globales del terreno, como la profundidad o la rugosidad global.

Todos ellos hacen referencia directa a características específicas del terreno y de los materiales que lo componen, y no a parámetros específicos de los métodos matemáticos internos (como el generador de ruido Perlin o algoritmo fractal).

En segundo lugar, los datos de entrada del generador de cuevas son (podemos ver la descripción detallada de todos ellos en la sección 6.2.3):

- Descriptores de los POIs: definen los diferentes tipos de POIs existentes.
- Relaciones de los POIs: determinan las afinidades entre POIs, así como las particularidades de la relación (distancia a la que se encuentran, sinuosidad del camino que los une, etc.)
- Parámetros globales: definen aspectos globales de todo el sistema de cuevas (como puede ser la escala, la sinuosidad global de todos los caminos, etc.).

Podemos ver que todos estos datos están relacionados directamente con el sistema de cuevas en sí y con los POIs que lo forman, y, al igual que en el generador anterior, no existen referencias a los métodos matemáticos utilizados internamente (en este caso el diagrama de Voronoi, la triangulación de Delaunay o el algoritmo de búsqueda de caminos).

En tercer lugar, los datos de entrada del generador de mazmorras son (podemos ver la descripción detallada de todos ellos en la sección 6.3.3):

- Definición de la mazmorra mediante grafos, donde los nodos representan unidades lógicas de esta (zonas o habitaciones) y los enlaces representan uniones entre dichas unidades (puertas, pasillos o escaleras).
- Parámetros globales: definen aspectos globales de toda la mazmorra, como el tamaño global de las habitaciones o la distancia entre diferentes zonas.

Al igual que en los dos generadores anteriores, todos los datos de entrada están relacionados directamente con el resultado (en este caso con la mazmorra y con su jugabilidad), obviando cuestiones técnicas. Además, al permitir definir familias de mazmorras con unos datos de entrada únicos, se disminuye el número de parámetros necesarios para generar múltiples mazmorras para poblar un mundo virtual.

Para finalizar, como vemos en la sección 4.2.2, el método de generación propuesto oculta todas sus peculiaridades y características técnicas, permitiendo acceder a la información volumétrica como si de un terreno voxelizado tradicional se tratase.

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de usabilidad, ya que todos los parámetros de entrada de nuestro método están relacionados exclusivamente con el resultado y no con aspectos técnicos.

Control

> **Requisito: 2**. El diseñador puede influir en todos los aspectos del resultado, pero no puede corregirlos.

Los tres métodos que componen el generador propuesto en esta tesis permiten definir en los datos de entrada todos aquellos aspectos que influyen en la generación de los resultados, es decir, todo el conocimiento para realizar el proceso de generación se encuentra en los datos de entrada y no en el proceso en sí mismo. Además, debido al requisito de velocidad, hemos evitado siempre pasos interactivos entre el diseñador y el método, por lo que es imposible alcanzar el nivel máximo de control.

El generador de terrenos permite controlar el resultado mediante la descripción de los materiales que componen el terreno (ver sección 5.3). Así, mediante la definición de sus afinidades, se establece cómo se combinan y cómo se distribuyen; y, mediante la definición de sus características físicas y modificadores (color, grosor de la capa, similitud con la capa anterior, etc.), se establece el aspecto que van a otorgar al terreno.

El generador de sistemas de cuevas tiene una filosofía similar, aunque en vez de definir materiales, define los tipos de cámaras subterráneas que va a tener la cueva, incluyendo sus afinidades y sus características (ver sección 6.2.3).

Debido a los requisitos de los elementos que crea, el generador de mazmorras permite al diseñador un mayor grado de detalle en el control del proceso de generación. Como podemos ver en la sección 6.3.3, el diseñador puede influir en todas las características que definen la estructura de una mazmorra, lo que le permite especificar con el

máximo grado de detalle la jugabilidad que va a presentar el resultado.

Respecto al sistema de representación propuesto, tal y como hemos visto en el capítulo 4, no presenta limitaciones en este aspecto, ya que no restringe de manera alguna las estructuras que el diseñador pueda tener intención de crear.

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de control. A pesar de ello, este aspecto es uno de los más complicados de determinar si alcanza o no el nivel establecido en los requisitos iniciales, ya que depende mucho de los deseos y del estilo de cada diseñador.

Ampliabilidad

> **Requisito: 2**. El métodos puede generar mundos virtuales de diferentes tipos mediante cambios en los parámetros de entrada o en la interacción con el diseñador.

Todos los métodos que conforman el generador propuesto en esta tesis basan su ampliabilidad en el hecho de que los datos de entrada no solo configuran los parámetros del algoritmo, sino que permiten definir aspectos centrales del proceso.

El generador de terrenos básicos fundamenta todo el proceso en los materiales que se utilizan, en sus afinidades y en sus características físicas. De esta manera, que el diseñador pueda definir cualquier tipo de material, permite generar terrenos de múltiples tipos y configuraciones. La figura 5.12 muestra un conjunto de terrenos creados mediante modificaciones simples de los datos de entrada.

Por su parte, al generador de cuevas le ocurre exactamente lo mismo. Este proceso basa los resultados en los POIs, elementos que, al igual que los materiales en el caso anterior, se definen por el diseñador, por lo que es posible crear cuevas intrincadas en las que el reto sea orientarse (por ejemplo, un laberinto), cuevas sencillas sin ramificaciones que conectan dos puntos de la superficie del terreno (como pueden ser canales para unir diferentes áreas o niveles lineales donde se dan una serie de acontecimientos encadenados) o cuevas simples con una única entrada y salida (como la cueva de un dragón en la que solamente hay una gran cámara con el dragón y otra más pequeña con el tesoro). Por un lado, la figura 6.9 muestra un conjunto de cuevas formadas por canales sencillos, por otro, la figura 6.1 muestra una cueva mucho más laberíntica. Desgraciadamente es muy difícil apreciar las diferencias jugables mediante imágenes.

El generador de mazmorras se basa completamente en los grafos que definen la estructura de la mazmorra, por lo que no existe ninguna limitación en su construcción

y la ampliabilidad es absoluta. Podemos ver algunos ejemplos de esta ampliabilidad en la figura 6.20, que presenta una mazmorra formada por una sola zona similar al plano de un edificio convencional; en la figura 6.25, que muestra una mazmorra con una jugabilidad simple; o en la figura 6.31, que presenta mazmorras mucho más complejas.

Finalmente, como vemos en el capítulo 4, la ampliabilidad del método se ve favorecida por el sistema de representación utilizado, ya que al igual que los métodos tradicionales de representación volumétricos no limita los elementos que puede almacenar.

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de ampliabilidad, ya que es posible obtener diferentes tipos de resultados mediante modificaciones de los datos de entrada.

Escalabilidad

Requisito: 1. El método puede generar mundos virtuales a diferentes escalas.

Primero, la escala de los terrenos se establece mediante un parámetro de entrada (ver sección 5.3.2), y, como vemos en la sección 5.4.2, afecta directamente al nivel de detalle en el proceso de generación, ya que se utiliza para determinar la frecuencia del ruido Perlin en la obtención de las alturas de cada capa y del flujo interno de material.

Segundo, el método que empleamos para seleccionar la escala en el generador de sistemas de cuevas es exactamente el mismo que para el generador de terrenos (ver sección 6.2.3.3). En este generador, como podemos ver en el primer paso de la sección 6.2.4.1, la escala afecta directamente al número de semillas con las que se construye el diagrama de Voronoi. Así, dependiendo de la escala, el tamaño de las celdas de Voronoi también se ve afectado, incidiendo directamente en el nivel de detalle de las cámaras subterráneas de la cueva, ya que diferentes tamaños de celda implican diferente número de *voxels* para representarlas.

Tercero, el generador de mazmorras es el que más limitado se encuentra en este aspecto debido a las características especiales de las mazmorras. A pesar de ello, el algoritmo basa la creación del resultado en una rejilla formada por unidades cuadradas relativas, por lo que permite al diseñador asignar un valor absoluto arbitrario a dicha unidad relativa, modificando la escala final de la mazmorra (como vemos en la sección 6.3.4).

Para finalizar, como vemos en el capítulo 4 el sistema de representación utilizado

requiere un número de elementos volumétricos (*voxels* en un sistema tradicional, limites en nuestro sistema) significativamente inferior a un sistema de representación tradicional. Podemos apreciar esta mejora en la tabla 4.1, donde se muestra que con un *chunk* con 20 *voxels* por cada lado se puede llegar a requerir, en el mejor de los casos, solamente el 10 % de los necesarios por un sistema tradicional. Este hecho permite representar terrenos con una gran cantidad de datos y a escalas mucho mayores que en los terrenos tradicionales.

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de escalabilidad, ya que nuestro generador es capaz de generar mundos virtuales a una escala arbitraria.

Estructura

> **Requisito: 2**. Los mundos virtuales obtenidos presentan estructuras reconocibles que se integran entre ellas.

La estructura de los resultados del generador de terrenos básicos viene dada por los siguientes factores:

- Los terrenos obtenidos por nuestro método presentan una estructura similar a los terrenos sedimentarios reales, ya que se encuentran formados por capas y vetas de materiales reconocibles. Podemos ver ejemplos de esto en la figura 5.13, en la que además podemos comparar dicha estructura con la de terrenos reales.

- Como vemos en la sección 5.4.2, la selección de los materiales para crear cada capa y cada veta se realiza utilizando las afinidades entre materiales definidas por el diseñador en los datos de entrada. Así, se mantiene la coherencia entre los materiales que forman cada capa y cada veta consecutiva.

- La generación de las alturas y la forma final de cada capa está influenciada por la forma de la superficie del terreno sobre la cual se construyen dicha capa (ver sección 5.4.2.1). Además, esta similitud viene dada por la configuración de materiales de la capa, definidos por el diseñador, por lo que es posible determinar qué materiales son más maleables y por tanto se adaptan mejor a la capa anterior, y cuáles no. Podemos ver un ejemplo de esto en la figura 5.12 y en las subfiguras 5.13(d), 5.13(e), 5.13(f) y 5.13(g), donde se muestra un corte transversal de los terrenos mediante el cual se pueden apreciar cómo las capas se integran entre sí.

Por su parte, el generador de sistemas de cuevas que proponemos mantiene la estructura en dos niveles diferentes:

- La estructura jugable se mantiene relacionando los diferentes puntos jugables de la cueva mediante sus afinidades e incompatibilidades definidas por el diseñador. De esta manera es posible establecer una serie de estructuras jugables mediante la afinidad entre diferentes POIs (por ejemplo, que exista una alta probabilidad de que después de un enemigo importante, exista un tesoro y no otro enemigo). Podemos ver un ejemplo de esto en la subfigura 6.3(d), en la que se han asignado los roles de los POIs creando la estructura jugable.
- La estructura visual se mantiene mediante la generación de galerías que relacionan cámaras subterráneas, integrando de esta manera toda la cueva en una única estructura. Podemos ver un ejemplo de esta estructura en un terreno bidimensional en la subfigura 6.3(d) y en un terreno tridimensional en la figura 6.9.

Por otro lado, nuestro generador de mazmorras, al igual que el de cuevas, mantiene una fuerte estructura visual y jugable en sus resultados:

- La estructura jugable se construye mediante la generación de la mazmorra a partir de un grafo que define las zonas, las habitaciones y los eventos jugables. De esta manera, estos elementos funcionan como unidades lógicas jugables, permitiendo establecer objetivos por capas para cada situación en la mazmorra (por ejemplo, un objetivo global —rescatar a la princesa al final de la mazmorra—, un objetivo de zona —encontrar el pasadizo a la siguiente zona— y un objetivo de habitación —superar a un enemigo—). Las figuras 6.26 y 6.31 muestran algunos ejemplos de esta estructura.
- La estructura visual, al igual que la jugable, viene dada por el grafo de entrada, ya que permite que la mazmorra esté generada por zonas, habitaciones y pasillos, estructurados y relacionados mediante puertas y escaleras con una ubicación lógica. Podemos ver un ejemplo de la estructura visual también en las figuras 6.26 y 6.31.

Además, como ocurre en apartados anteriores, el hecho de que el método propuesto para representar los terrenos volumétricos no restrinja de manera alguna el tipo ni las características de las estructuras que un terreno puede contener, permite que el nivel de estructura no se vea limitado por él (ver capítulo 4).

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de estructura, ya que todos los elementos obtenidos por nuestro generador de mundos virtuales son reconocibles y se encuentran integradas entre sí. Además, también se cumple el requisito adicional que dice que los terrenos deben estar formados por capas de materiales.

Interés

> **Requisito: 2**. Los mundos virtuales obtenidos presentan estructuras explorables y, además, motivan dicha exploración de alguna manera.

En primer lugar, los terrenos creados con nuestro método son, por definición, absolutamente explorables, pudiendo llegar a ser incluso teóricamente infinitos (ver sección 5.4.4). Además, la exploración se ve motivada por la distribución de los recursos jugables en vetas minerales más o menos difíciles de acceder. En la figura 5.12 podemos ver algunos terrenos generados por nuestro método con vetas de mineral a diferentes profundidades.

En segundo lugar, las cuevas, al igual que los terrenos, son explorables por definición. Además, la exploración se ve motivada por la jugabilidad con la que se construye la cueva (por ejemplo, dando recompensas a los jugadores después de superar a enemigos importantes o escondiendo en lo más profundo alguna cámara especialmente interesante para el desarrollo del juego). La subfigura 6.3(d) muestra los POIs que pueblan una cueva y que incitan al jugador a adentrarse en ella para buscar las recompensas.

A continuación, el generador de mazmorras propuesto en esta investigación permite al diseñador determinar de una manera estricta la jugabilidad de la mazmorra, por lo que el nivel de interés de las mazmorras resultantes es similar al nivel de interés de las mazmorras creadas a mano con un método tradicional. Las figuras 6.26 y 6.31 muestran algunos ejemplos, donde se puede ver cómo está estructurada la jugabilidad en capas (el objetivo global en todas ellas es alcanzar la cámara del tesoro, aunque hay objetivos parciales, como encontrar las llaves que abren sus correspondientes puertas o superar a los diferentes enemigos). Además, la capacidad de creación de familias de mazmorras permite que un mundo virtual que contenga múltiples mazmorras del mismo tipo siga manteniendo el interés, ya que es posible determinar libremente sus diferencias, tanto jugables como estructurales.

Como vemos, el generador de terreno también contribuye en el interés global, aunque es en la capa jugable (generador de cuevas y generador de mazmorras) donde recae

la mayor parte del peso para obtener un nivel de interés adecuado en el mundo virtual.

Para finalizar, al igual que en apartados anteriores, el sistema de representación propuesto permite que los terrenos estén formados por cualquier tipo de estructura y elemento (ver capítulo 4), por lo que no limita el nivel interés del mundo virtual.

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de interés, aunque este es altamente dependiente de factores como el objetivo del mundo virtual o la habilidad del diseñador para idear mundos virtuales interesantes. Además, también se cumplen los requisitos adicionales en los que se especifica que el mundo virtual debe contener estructuras jugables como cuevas o mazmorras y que el terreno debe contener recursos jugables distribuidos como vetas de mineral.

Velocidad

> **Requisito: 1**. El resultado se genera en tiempo de ejecución en un paso previo a la interacción con el jugador. Método híbrido entre *offline* y *online*.

Como podemos ver en la sección 5.4.4, la generación de terrenos puede ejecutarse de manera *online*. Además, podemos ver el tiempo que tarda en realizar todo el proceso en la tabla 5.3.

Por su parte, tal y como vemos en la sección 6.2, la generación de cuevas es un proceso que se ejecuta de una manera híbrida. A pesar de ello, existen varios motivos que nos permiten pensar que es un método capaz de ejecutarse de manera *online*: por un lado, analizando los pasos requeridos para la generación de las cuevas así como las características indeterminadas de su estructura, podemos concluir que son un elemento propenso a ser construido mientras se va explorando; por otro lado, en la sección 6.2.5.1 podemos comprobar que aunque la ejecución del proceso tarda demasiado tiempo (ver tabla 6.6), el paso que más tiempo requiere es la generación del diagrama de Voronoi y la triangulación de Delaunay (ver tabla 6.7). Así, parece posible adaptar el método para que siga creando los diagramas de manera híbrida, pero que la cueva la construya de manera *online*.

Para finalizar, el método de generación de mazmorras, debido a las restrictivas características que deben presentar sus resultados (ver sección 6.1.2), se debe ejecutar siempre de manera híbrida.

Por ello, podemos afirmar, sin requerir experimentos adicionales, que nuestro mé-

todo es capaz de ejecutarse de manera híbrida, por lo que el objetivo propuesto en los requisitos para la velocidad de ejecución se cumple.

Realismo

> **Requisito: 2**. Los resultados se basan en nuestra realidad, pero sin llegar a simular fenómenos naturales.

Ninguno de los métodos propuestos en esta investigación tiene como objetivo una simulación realista de fenómenos naturales, aunque sí se basan en elementos reales para generar los resultados.

En primer lugar, podemos comprobar el realismo de los terrenos obtenidos por nuestro método en los siguientes aspectos:

- Los terrenos obtenidos con nuestro generado están basados en capas y vetas de materiales, al igual que los suelos sedimentarios naturales. Podemos observar esta similitud en la sección 5.5.3, donde realizamos una comparación entre nuestros terrenos y cortes de terreno real.
- Las capas que forman el terreno basan su topología en el generador de ruido Perlin (ver sección 5.2.1), tradicionalmente utilizado para simular elementos naturales realistas [MKM89, SdKT+09] (ejemplos en las figuras 5.12 y 5.5.3). Además, también utilizamos este generador para obtener el flujo de material de cada capa, tal y como podemos observar en la figura 5.3.
- Las capas y vetas de los terrenos están formadas por una colección de materiales con diferentes concentraciones, generada mediante un método probabilístico definido por las afinidades de los materiales cercanos. Por ejemplo, la subfigura 5.12(b) muestra un terreno en el que la capa amarilla está formada por la mezcla de otros dos materiales, a pesar de que en los datos de entrada utilizados para generar este terreno no se incluye un material de color amarillo. Este hecho simula las composiciones de los terrenos reales, en los que las diferentes capas sedimentarias suelen incluir mezclas de diferentes materiales.
- Los terrenos obtenidos con nuestro método realizan la transición entre capas consecutivas de manera progresiva, evitando fronteras muy marcadas. Además, el diseñador puede determinar mediante los parámetros de materiales cómo se lleva a cabo esta transición (por ejemplo, una capa de arena se mezclará con mayor

facilidad que una de roca). Podemos ver varios ejemplos en las figuras 5.12 y 5.13.

En segundo lugar, el realismo de las cuevas obtenidas con nuestro método se da por los siguientes aspectos:

- La generación de los sistemas de cuevas contempla la ramificación de los caminos, por lo que es posible obtener complejos sistemas de galerías subterráneas evitando dar al resultado una apariencia demasiado lineal y artificial. Podemos ver un ejemplo sencillo de cómo se ha ramificado una cueva en la subfigura 6.3(c) y una cueva más compleja en un entorno tridimensional en la figura 6.1.

- Todas las cámaras y galerías subterráneas obtenidas con nuestro generador están construidas sobre un diagrama de Voronoi tridimensional con sus semillas distribuidas aleatoriamente, por lo que sus celdas presentan una estructura irregular (como podemos ver en la subfigura 6.2(a)) dando naturalidad a la apariencia del sistema de cuevas.

- La generación de las galerías se realiza teniendo en cuenta la sinuosidad (ver sección 6.2.4.4). Así, tal y como observamos en las figuras 6.1 y 6.9, el proceso evita generar galerías completamente rectas para dar naturalidad al resultado.

- La estructura jugable de la cueva organiza los eventos mediante las afinidades establecidas por el diseñador, por lo que tiene tanto realismo como el diseñador desee. Por ejemplo, si quiere diseñar una cueva que represente la guarida de unos ladrones, el diseñador puede simularla creando un puesto de guardia justo en la entrada de la cueva, o situando la sala del tesoro en lo más profundo. La subfigura 6.3(d) muestra una cueva con los POIs asignados, en la que para llegar desde la entrada al tesoro es obligatorio pasar por un enemigo.

En tercer lugar, podemos comprobar el realismo obtenido por el generador de mazmorras que proponemos en esta investigación mediante los siguientes aspectos:

- Las mazmorras intentan simular estructuras manufacturadas tales como minas o ciudades subterráneas. Para ello, el generador estructura los resultados en agrupaciones de habitaciones interconectadas, relacionándolas con otras agrupaciones mediante corredores y escaleras (ver figura 6.31).

- Al igual que la de las cuevas, la estructura jugable de las mazmorras presenta todo el realismo que el diseñador quiera, ya que es este el que distribuye los eventos por las habitaciones y define el rol que cumple cada una. Podemos ver algunas mazmorras con los eventos distribuidos en las figuras 6.26 y 6.31.

Finalmente, al igual que en apartados anteriores, el sistema de representación propuesto favorece el realismo de los terrenos generados debido a que permite almacenar una mayor cantidad de información que los sistemas tradicionales (ver sección 4.3), y por tanto representar un nivel de detalle mayor. Además, al no limitar los tipos de elementos que puede almacenar, permite que los terrenos estén formados por cualquier tipo de estructura y elemento, por lo que el realismo solo se ve afectado por la capacidad del método de generación utilizado.

Como podemos ver, todos los aspectos presentados en este apartado indican que se cumple el requisito de realismo.

7.2 Comparación con otros generadores procedurales

En esta sección se realiza una comparación entre el método propuesto y otros generadores procedurales existentes actualmente, utilizando los aspectos críticos y sus respectivas métricas propuestas en el capítulo 3.

Ninguno de los métodos con los que comparamos nuestro generador realiza una evaluación que determine objetivamente su nivel en los diferentes aspectos críticos de los generadores procedurales. Debido a ello, para establecer el valor de cada uno de estos aspectos de cada método, seguimos los siguientes pasos:

1. Si la publicación científica donde se propone el método hace referencia explícita al aspecto que queremos evaluar (por ejemplo, mide la velocidad de ejecución o trata el interés jugable del resultado de algún modo), le asignamos el valor de nuestra métrica equivalente a lo que el artículo dice sobre dicho aspecto. Si no existe ninguna referencia explícita, continuamos con el siguiente paso.

2. Buscamos en la publicación científica donde se propone el método alguna referencia implícita al aspecto que queremos evaluar (por ejemplo, no hace referencia directa a la ampliabilidad del método, pero muestra el proceso para generar múltiples tipos de resultados mediante su método), analizamos el método (de la misma manera que lo hacemos para nuestro método en la sección 7.1), asignándole un valor de nuestra métrica dependiendo del resultado de dicho análisis. Si

no existe ninguna referencia, ni explícita ni implícita, dejamos sin evaluar ese en para el generador en cuestión.

Si el método es industrial y no lleva una publicación científica asociada, el proceso de evaluación se limita a llevar a cabo un análisis como el realizado para nuestro método en la sección 7.1 para cada uno de los aspectos.

Este proceso permite asignar un valor de nuestras métricas a cada aspecto crítico de todos los métodos con los que comparamos nuestro generador[1], pero, como hemos visto en la sección 7.1, este valor es solo una estimación. A pesar ello, utilizamos estos valores para realizar una primera aproximación a la evaluación del método que proponemos en esta investigación y ver su posible potencial respecto a otros generadores procedurales existentes.

Los métodos con los que realizamos la comparación son aquellos del estado del arte que, de algún modo, comparten objetivo con nuestra investigación. Para ello, dividimos la comparación en tres apartados diferentes: generación de terreno, generación de cuevas y generación de mazmorras. Para cada uno de estos apartados seleccionamos y evaluamos los métodos más importantes, que son los siguientes:

Para la generación de terrenos, seleccionamos los generadores de terrenos fractales (sección 2.2.1.1) —que debido a sus similares características se han agrupado como un solo generador—, y los generadores de terrenos evolutivos (sección 2.2.1.2) —aunque obviamos el método de Walsh y Gade [WG10] debido a que su objetivo no es generar terrenos, sino utilizar un generador de terrenos para obtener imágenes realistas no interactivas—. Además, agregamos los generadores de aquellos videojuegos con mundos virtuales volumétricos procedurales más conocidos actualmente, que son *Minecraft* [Moj11], *Terraria* [Re-11] y *Starbound* [Chu13], aunque en este apartado solamente comparamos la generación del terreno de dichos métodos.

Por su parte, para la generación de cuevas, seleccionamos todos los generadores de este tipo de contenido presentes en el estado del arte (sección 2.2.2). Además, como en el punto anterior, se han seleccionado los generadores de aquellos videojuegos con mundos virtuales volumétricos procedurales más conocidos actualmente, aunque en este apartado solamente comparamos la generación de las cuevas de dichos métodos.

Finalmente, para la generación de mazmorras seleccionamos aquellos métodos del estado del arte cuyos objetivos son la generación de espacios jugables que se asemejen

[1]Siempre que sea imposible asignar un valor a un aspecto de un generador, en la tabla se marcará mediante un guion (-).

	Innovación	Usabilidad	Control	Ampliabilidad	Escalabilidad
Nuestro método	2	1	2	2	1
Minecraft	1	-	-	-	0
Terraria	1	-	-	-	0
Starbound	2	-	-	-	0
Terrenos fractales	1	0	1	0	1
Ong *et al.*	1	1	2	2	1
Ashlock *et al.*	2	1	2	2	1
Frade *et al.*	1	0	3	0	1
Togelius *et al.*	1	0	2	0	0
Raffe *et al.*	0	1	2	2	1

Tabla 7.1: Comparación con otros generadores de terreno (aspectos de diseñador).

a este tipo de contenido (Dormans [Dor10] y Sorensons y Pasquier [SP10]). Además, como en el punto anterior, se han seleccionado los generadores de aquellos videojuegos con mundos virtuales volumétricos procedurales más conocidos actualmente, aunque en este apartado solamente comparamos la generación de las mazmorras de dichos métodos.

Las tablas 7.1 y 7.2 muestran los resultados de la comparación de los generadores de terreno, las tablas 7.3 y 7.4 los resultados de la comparación de los generadores de cuevas, y las tablas 7.5 y 7.6 los resultados de la comparación de los generadores de mazmorras.

Como podemos ver, existen generadores que superan en algunos aspectos a nuestro método, pero, a pesar de ello, el nuestro es el único que alcanza completamente los requisitos establecidos en la sección 3.2.3.

	Estructura	Interés	Velocidad	Realismo
Nuestro método	2	2	1	2
Minecraft	1	2	2	1
Terraria	2	2	1	1
Starbound	2	2	1	1
Terrenos fractales	2	0	2	2
Ong *et al.*	1	1	1	-
Ashlock *et al.*	1	1	1	-
Frade *et al.*	2	1	1	-
Togelius *et al.*	1	2	1	-
Raffe textitet al.	1	2	0	-

Tabla 7.2: Comparación con otros generadores de terreno (aspectos de usuario).

	Innovación	Usabilidad	Control	Ampliabilidad	Escalabilidad
Nuestro método	2	1	2	2	1
Minecraft	2	-	-	-	0
Terraria	2	-	-	-	0
Starbound	2	-	-	-	0
Boggus y Crawfis	1	-	-	0	1
Peytavie *et al.*	2	0	3	2	1
Johnson *et al.*	2	0	1	0	1
Juncheng *et al.*	1	0	2	0	1

Tabla 7.3: Comparación con otros generadores de cuevas (aspectos de diseñador).

	Estructura	Interés	Velocidad	Realismo
Nuestro método	2	2	1	2
Minecraft	2	1	2	2
Terraria	2	2	1	2
Starbound	2	1	1	2
Boggus y Crawfis	2	1	-	3
Peytavie *et al.*	2	1	0	3
Johnson *et al.*	0	1	2	1
Juncheng *et al.*	2	1	-	2

Tabla 7.4: Comparación con otros generadores de cuevas (aspectos de usuario).

	Innovación	Usabilidad	Control	Ampliabilidad	Escalabilidad
Nuestro método	2	1	2	2	1
Minecraft	1	-	-	-	0
Terraria	1	-	-	-	0
Starbound	1	-	-	-	0
Dormans	1	0	2	2	-
Sorensons y Pasquier	1	1	1	2	-

Tabla 7.5: Comparación con otros generadores de mazmorras (aspectos de diseñador).

	Estructura	Interés	Velocidad	Realismo
Nuestro método	2	2	1	2
Minecraft	2	0	2	2
Terraria	2	1	1	2
Starbound	2	1	1	2
Dormans	2	2	-	-
Sorensons y Pasquier	1	2	0	-

Tabla 7.6: Comparación con otros generadores de mazmorras (aspectos de usuario).

7.3 Debilidades y fortalezas del método propuesto

Una vez comparados los resultados del generador propuesto en esta tesis con los requisitos recopilados en la etapa inicial de la investigación, podemos extraer las fortalezas y las debilidades de nuestro método.

Debilidades

Por un lado, nuestro generador presenta los siguientes puntos débiles:

- Debido a que la mayor parte de videojuegos que utilizan terrenos volumétricos utilizan terrenos basados en capas, nuestro generador basa todo el proceso en la creación de terrenos sedimentarios. Por ello, es muy complicado establecer unos parámetros de entrada para obtener otros tipos de terrenos. A pesar de esto, es posible mejorar el generador para que sea capaz de crear terrenos de otros tipos sustituyendo el proceso de generación de capas por otras técnicas dedicadas a obtener volúmenes, como la generación de nubes de puntos más o menos aleatorias y el cálculo de su envolvente convexa.
- Los terrenos obtenidos están formados por capas cuya forma se determina mediante un generador de ruido Perlin al que se le aplican diferentes modificadores, lo que provoca que la superficie de la capa superior del terreno tenga también estas características, por lo que su aspecto es bastante simple. Este problema puede solucionarse aplicando un método de simulación de la erosión sobre el terreno ya generado.
- Como hemos visto, el tiempo requerido para la generación de las cuevas es alto, aunque puede realizarse en dos pasos amortiguando el efecto de este problema.
- Los sistemas de cuevas obtenidos con el generador propuesto tienen una alta variabilidad tanto en la configuración de la cueva (número de ramificaciones, posición de los POIs, etc.) como en el tiempo que tardan en generarla. Este efecto puede evitarse mediante una configuración cuidadosa y suficientemente restrictiva de los datos de entrada.
- Cuando el grafo que define una mazmorras es demasiado complejo, el método de generación propuesto puede ser incapaz de generarla. Por ejemplo, si un grafo define una zona con un número elevado de habitaciones todas ellas interconectadas, es posible que no exista una solución válida.

Podemos comprobar que de las cinco debilidades identificadas, las tres primeras son enmendables, mientras que las otras dos dependen de los parámetros de entrada que establece el diseñador.

Fortalezas

Por otro lado, nuestro método presenta los siguientes puntos fuertes:

- Todos los aspectos analizados indican que nuestro método supera los requisitos establecidos inicialmente, a pesar de que la evaluación del método no es definitiva. Así, todo indica que el método propuesto es perfectamente válido para ser utilizado en la industria.

- La alta capacidad de personalización de los resultados que permite el método a los diseñadores hace que este sea útil para multitud de propósitos. Por ejemplo, el generador que proponemos puede utilizarse para diferentes géneros de videojuegos y aplicaciones interactivas, abarcando desde juegos de exploración y supervivencia, hasta simuladores militares. En esencia, cualquier tipo de juego que requiera terrenos de grandes proporciones puede utilizar el generador propuesto en esta investigación.

- El proceso que utiliza nuestro generador de terrenos para distribuir los recursos en vetas de mineral encaja perfectamente con los videojuegos existentes en la industria que basan su jugabilidad en la supervivencia y en la recolección de recursos.

- Nuestro generador de cuevas define la jugabilidad de la cueva utilizando POIs, permitiendo definir cualquier tipo de jugabilidad. Este acercamiento, además, encaja perfectamente con todos aquellos videojuegos comerciales que basan su jugabilidad en la exploración.

- El generador de mazmorras de nuestro método permite un grado de control tan alto, que posibilita definir y generar mazmorras mucho más complejas que las que se pueden ver en videojuegos comerciales con mundos virtuales procedurales (*Minecraft* [Moj11], *Terraria* [Re-11], o *Starbound* [Chu13], por ejemplo), y está complejidad se acerca mucho a la de las mazmorras de juegos creados de la manera tradicional (como puede ser *The Elder Scrolls V: Skyrim* [Bet11]).

7.4 Validación de la hipótesis inicial

Volviendo a la hipótesis desde la que parte esta investigación (ver sección 1.4), podemos comprobar que está compuesta por dos partes diferenciadas.

> «*Es posible, empleando una combinación de técnicas espaciales y de inteligencia artificial, generar mediante métodos no supervisados mundos virtuales volumétricos que cumplan todos los requisitos, tanto explícitos como implícitos, propuestos tanto por la comunidad científica como por el entorno industrial. Además, es posible sintetizar este conjunto de requisitos formalmente, obteniendo métricas objetivas para evaluarlos.*»

Por un lado, tenemos la identificación y síntesis de los aspectos críticos de los generadores procedurales. Este proceso los hemos realizado con éxito, recopilando dichos aspectos, tanto explícitos como implícito presentes en la industria y en la literatura científica, y proponiendo una serie de métricas para evaluarlos con una mayor objetividad que con la variedad de métodos que se utilizan actualmente para este propósito. Además, hemos analizado los generadores procedurales más utilizados y se hemos establecido unos requisitos definiendo el nivel mínimo que cada aspecto debe tener utilizando las métricas desarrolladas.

Hasta donde llega nuestro conocimiento, esta investigación es la primera en dar un paso hacia la estandarización del método y los criterios de evaluación de los generadores procedurales.

Por otro lado, hemos desarrollado un método de generación de mundos virtuales volumétricos utilizando como guía los requisitos propuestos previamente. Para validar completamente que todos los aspectos críticos cumplen estos requisitos, es necesaria una evaluación exhaustiva mediante grupos de usuarios y diseñadores, y una implementación el método de generación en un videojuego real. A pesar de ello, como hemos visto previamente, todos los resultados señalan que el método propuesto supera los requisitos iniciales.

Atendiendo a los objetivos definidos al comenzar esta investigación, podemos afirmar que hemos cumplido los cuatro primeros objetivos operacionales (identificación de los aspectos críticos presentes en la literatura; identificación de los aspectos críticos presentes en la industria; propuesta de un conjunto de métricas y requisitos; y diseño de un generador que atienda a dichos requisitos), mientras que para el quinto (evaluación del método mediante los requisitos) solamente hemos realizado una primera aproximación.

Así, solamente podemos afirmar que, de los dos objetivos específicos (identificación y síntesis de los requisitos; y desarrollo y evaluación de un generador procedural que los cumpla) hemos cumplido el primero, mientras que el segundo queda parcialmente desarrollado.

En conclusión, podemos determinar que este trabajo de investigación ha alcanzado un éxito parcial, ya que aunque no podemos afirmar de una manera rotunda que la hipótesis ha sido validada hasta que se lleven a cabo los experimentos propuestos, todos los resultados obtenidos en la evaluación cualitativa realizada indican que el método supera los requisitos y que se ha alcanzo el objetivo inicial.

7.5 Contribuciones

Las principales contribuciones realizadas en esta investigación son las siguientes:

- Hemos definido un conjunto de aspectos críticos para los generadores procedurales de mundos virtuales mediante un análisis de la literatura y de los generadores industriales. Además, hemos propuesto un conjunto de métricas para evaluarlos.
- Utilizando tanto los aspectos críticos como las métricas propuestas, hemos definido un conjunto de requisitos que todo generador procedural de mundos virtuales debe cumplir para ser válido directamente para la industria.
- Hemos propuesto un método de representación de terrenos volumétricos que mejora la cantidad de espacio necesario para almacenar un terreno frente a los terrenos tradicionales basados en *voxels*.
- Hemos propuesto un nuevo método de generación de terrenos volumétricos basado en terrenos sedimentarios en el que todos sus resultados y aspectos indican que alcanza los requisitos iniciales.
- Además, hemos propuesto un nuevo método de generación de sistemas de cuevas basado en diagramas de Voronoi, en el cual el diseñador puede definir que tipos de puntos jugables van a formar la cueva y como se relacionan entre sí. Al igual que el generador de terrenos, todos los resultados y aspectos de este generador indican que también alcanza los requisitos iniciales.
- Finalmente, también hemos propuesto un nuevo método de generador de mazmorras basado en un método de división de estancias mediante técnicas de presión, en el cual se obtienen familias de mazmorras con una variabilidad controlada y

con una jugabilidad estricta definida por el diseñador. Al igual que el generador de terrenos, todos los resultados y aspectos de este generador indican que también alcanza los requisitos iniciales.

De esta manera, no solo hemos propuesto un generador procedural de mundos virtuales, sino que hasta donde llega nuestro conocimiento, hemos sido los primeros en dar un paso hacia la estandarización y organización de los criterios de evaluación de este tipo de generadores procedurales.

7.6 Trabajo futuro

Teniendo en cuenta lo expuesto en secciones anteriores de este capítulo, proponemos tres líneas de investigación diferentes para avanzar en este campo partiendo desde el punto en el que lo deja esta tesis: la evaluación completa del método, el aumento del detalle en los resultados y la mejora de la velocidad de generación.

Evaluación completa

Como hemos expuesto anteriormente, la validación total del método propuesto requiere un profundo proceso de evaluación.

Dependiendo del conjunto de aspectos a evaluar los pasos a seguir son diferentes.

Por un lado, la evaluación de los aspectos de diseñador (sección 3.1.1) requiere una serie de experimentos orientados al proceso de diseño de mundos virtuales. Nuestra propuesta es la realización de una serie de sesiones de trabajo de diseñadores utilizando el método propuesto en esta tesis y evaluando después el proceso mediante una encuesta.

Por otro lado, para evaluar el conjunto de aspectos de usuario (sección 3.1.2) proponemos su evaluación mediante una aplicación gráfica interactiva. Para ello, es necesario implementar el método propuesto en un videojuego y realizar una serie de sesiones con diferentes tipos de jugadores, para posteriormente, evaluar los aspectos de usuario midiendo mediante una encuesta la experiencia sufrida por los participantes.

El desarrollo de esta línea de trabajo, además de evaluar nuestro método de una manera definitiva, permitiría realizar una comparación más rigurosa entre diferentes métodos.

Aumento del detalle

El método propuesto en esta tesis genera mundos virtuales volumétricos, pero no ahonda en el detalle de cada zona. Debido a ello, proponemos una línea de trabajo futuro que se encargue de profundizar en este aspecto.

Algunos de los elementos susceptibles de ser abordados por esta línea son:

- La superficie del terreno obtenida por el método propuesto es muy simple y no presenta elementos tales como ríos o zonas erosionadas. Para ello, proponemos investigar una posible aplicación de alguno de los métodos que simulan la erosión térmica y fluvial presentados en la sección 2.2.1. Uno de los mayores retos en esta línea de investigación es conseguir un método de simulación de la erosión que no limite la velocidad de generación del proceso en general.
- Las cuevas generadas por nuestro método presentan una apariencia natural debido a las características desiguales de los diagramas de Voronoi. A pesar de ello, el realismo en el interior de los canales se ve limitado por la falta de elementos característicos como estalactitas y estalagmitas. Nuestra propuesta en esta línea de investigación es la búsqueda de un método que simule este tipo de construcciones y no lastre la velocidad de ejecución del algoritmo completo.

Mejora de la velocidad de generación

Como hemos visto en capítulos anteriores, el método puede ejecutarse de manera híbrida (valor **1** en el aspecto de velocidad de generación definido en la sección 3.1.2). A pesar de ello, algunos de los pasos definidos presentan indicios de tener un gran margen de mejora en su tiempo de ejecución.

Además, es posible utilizar diferentes métodos para generar el diagrama de Voronoi y la triangulación de Delaunay en la generación de cuevas, y para obtener los caminos en la generación de mazmorras, por lo que esos pasos son muy susceptibles de mejorar su tiempo probando algoritmos más eficientes que los empleados.

7.7 Consideraciones finales

El rápido desarrollo de la industria de los videojuegos y otras aplicaciones gráficas interactivas hace que sea imposible alcanzar el método definitivo para la generación pro-

cedural de mundos virtuales. El aumento exponencial de la cantidad de detalle requerido implica que este tipo generadores deban estar continuamente reinventándose para adaptarse a las nuevas tecnologías y tendencias. Además, debido a las características especiales que ofrece el contenido procedural a la jugabilidad de las aplicaciones interactivas, se apliquen cada vez en más géneros diferentes, aumentando en gran medida la complejidad del problema.

Debido a ello, el estudio de nuevos métodos y técnicas que permitan generar proceduralmente tanto mundos virtuales como cualquier otro elemento debe discurrir siempre en paralelo al crecimiento técnico y social de las aplicaciones gráficas interactivas.

Por último, como hemos visto, no existen unos criterios estándares ni un método definitivo para la evaluación de este tipo de generadores, por lo que la continuidad del trabajo realizado en esta investigación en este aspecto es crucial para el sector.

«Cake, and grief counselling, will be available at the conclusion of the test.»

GLaDOS (*Portal*, Valve, 2007)

CAPÍTULO

8

Results analysis and conclusions

We have proposed a new method for procedural volumetric virtual worlds generation (chapters 4, 5 y 6). Additionally, we have analysed the state of the art of the procedural generation, extracting a set of metrics and requirements procedural generators should meet.

This chapter analyses the results obtained by the proposed methods, extracting some conclusions of the research. It is structured as follows: First, the section 8.1 evaluates the success of the method using the requirements and the metrics defined in the chapter 3, and the section 8.2 compares our method against other generators. Second, the section 8.3 enumerates the strengths and weaknesses of our method. Next, section 8.4 compares the final results against the initial hypothesis, and the section 8.5 enumerates the main contributions of our research. Finally, the section 8.6 proposes a set of lines for future work, and the section 8.7 presents a set of final considerations extracted from the knowledge obtained during the research.

8.1 Comparison against the initial requirements

In this section, we analyse the proposed method using the metrics and the requirements defined in the chapter 3.

The evaluation of those aspects requires the following experiments: On the one hand, in order to evaluate the designer oriented aspects, a evaluation by a group of designers is required. On the other hand, in order to evaluate the user oriented aspect, the method should be implemented in a real video game, and it should be evaluated by a group of players.

In both cases the experiments are out of the scope of this research, and the resources they require are beyond our capabilities. Due to this fact, we perform a first approach with a direct analysis of the method, leaving those experiments for the next step in this research.

Innovation

> **Requirement: 2**. Obtained results present unexpected elements with significant differences between them.

The terrain and cave generators proposed in this research determine the element to construct next and its characteristics using a random process based on the affinities with other elements already generated.

On the one hand, the features of the terrain generation method that enhance the innovation are:

- The terrains are constructed by material layers and mineral veins (see figure 5.12). As shown in section 5.4.2, the element which is going to be constructed next (a layer or a vein) is selected evaluating a probability based on the density of veins desired by the designer. So, it is impossible to predict how many layers and veins are going to be created and their distribution.

  The experiments have been performed with a 0.5 density of veins (see table 5.2), and the results show that the average number of layers created is 4.8 and the average number of veins created is 1.7. We can check that the terrains have a variable number of layers and veins, because the results present decimal numerals.

- The layers and the veins are formed by a set of materials with different concentrations. As shown in section 5.4.2, this collection is determined by a probabilistic

method based on the affinity between nearby materials. Figure 5.12 shows a collection of terrains whose layers are formed by sets of materials. For example, the yellow layer of the terrain of the subfigure 5.12(b) is formed by the combination of other two materials, because there is no material defined in the input data with this colour.

Unless the input data was too restrictive, it is impossible to predict the configuration of the materials of the layers and veins, so, the generator creates unexpected elements not necessarily defined in the input data.

- As shown in the section 5.4.2, the final shape of each layer is defined by the surface of the previous layer and by a Perlin noise generator, and the section 5.2.1 shows that those generators obtain random noise, so, it is impossible to predict accurately the final results.

- The final shape of each vein is generated by a fractal algorithm with a high level of randomness (the process is explained in section 5.4.2.2). The subfigure 5.9(c) shows a fractal shape obtained with this method.

On the other hand, the features of the cave system generation method that enhance the innovation are:

- The section 6.2.4 shows that the cave systems are based on a graph whose nodes represent points with a playable interest (POI) (see figure 6.3(c)). The method decides which type of POI is going to be constructed next by a random procedure, using probabilities based on the affinities and incompatibilities with other type of POIs (see section 6.2.4.2).

  As in the layers material selection process, unless the input data was too restrictive, it is impossible to predict the structure of the caves, so, the generator creates unexpected elements.

- The characteristics of the POIs (depth, size of the underground chamber, distance to other POIs, number of galleries between other POIs, etc.) are randomly obtained, using the range defined by the designer (see section 6.2.4.3).

- Section 6.2.4.4 shows that the generation of the underground galleries contains a level of sinuosity defined by the designer, adding randomness to the pathfinding algorithm and making the results impossible to predict. We can see a cave with a high level of sinuosity in the subfigure 6.9(b).

- As shown in section 6.2.4.3, the shape of each underground chamber is determined by the cell of the Voronoi diagram (see figure 6.2(a)), which is determined by the initial distribution of the seeds (white noise distribution). So, the shape of the cave is completely unpredictable. In the figure 6.3 we can see an example of the randomness level of the underground chambers.
- As shown at the first point, the caves are constructed expanding a graph formed by POIs using their affinities. Due to this fact, it is impossible to predict the final shape of the cave.

Finally, the innovation of the dungeon generator is limited by the high restrictions of this type of content. In spite of that, the introduction of probabilities for the existence of some elements of the dungeon enables the designer to define the level of uncertainty of the result. Moreover, the zone and room locating procedure and the room growing method, generate unpredictable results.

Additionally, the proposed volumetric terrain representation method does not limit the type of structures or their features, enabling the generator to create any type of construction.

Due to all of this, we can guess that the virtual worlds obtained with our method presents unexpected elements with significant differences between them, meeting this requirement.

Usability

> **Requirement: 1**. Input parameters are exclusively related with the result, avoiding technical aspects.

The input data for the terrain generator is formed by the following parameters (view section 5.3):

- Descriptors of the materials.
- Modifiers of the materials for the layers and veins.
- Affinities between materials.
- Global aspects of the terrain.

We can observe that all of them are directly related to the terrain or to the materials, avoiding references to the technical method used.

On another front, the input data of the cave generator is formed by the following parameters (view section 6.2.3):

- Descriptors of the POIs.
- Relationships of the POIs.
- Global aspects of the caves.

We can observe that they are related to the cave system, avoiding technical concepts and references to the used methods (Voronoi diagram, Delaunay triangulation, etc.).

Finally, the input data of the dungeon generator is formed by the following parameters (view section 6.3.3):

- Definition of the dungeon structure with graphs, where the nodes are zones or rooms and the links represents connections between them.
- Global aspects of the dungeon.

As in the previous two cases, we can observe that all the parameters are related with the dungeon and its playability, avoiding technical questions.

Additionally, the proposed representation method hides its technical peculiarities, allowing the access to the volumetric data as in the traditional voxelized volumetric terrains.

Due to all of this, we can guess that the proposed method meets the usability requirement.

Control

> **Requirement: 2**. The designer has influence over all the aspects of the results, but it cannot correct them interactively.

The proposed method enables the designer to define in the input data all the aspects with influence in the generation process. That is, all the knowledge of the result is

contained in the input data and not in the method itself. Moreover, due to the speed requirement, the method avoids interactive steps in the process.

Terrain generator enables the designer to control the result with the material descriptions (see section 5.3): their affinities determine how they are combined and their characteristics (colour, similarity with the last layer, etc.) determine how they affect the terrain.

On the other hand, the cave system generator enables the designer to control the results in a similar way: the designer determines all the desired features for the cave defining the POI types and their affinities (see section 6.2.3).

Finally, due to the requirements of the dungeons, the dungeon generator enables the designer to have a more detailed control over the results. As shown in section 6.3.3, the designer has influence over all the features which define the dungeon playability.

Additionally, the data representation method can store any type of element without restriction, so, it does not restrict in any way the structures that the designer may have intended to create.

In spite of that, this requirement is one of the most difficult to guess, because it is strongly dependent of the intention and the style of each designer.

Expandability

> **Requirement: 2**. The method can generate different types of results with simple changes in the input data.

The expandability of the generator is based on the fact that it allows to define core aspects of the results in their input parameters.

On the one hand, the terrain generation process is based on the materials, which are completely defined by the designer in the input data. We can see an example of the expandability of this process in the figure 5.12.

On the other hand, the cave generation process is based on the POI types, which are defined by the designer. This fact allows to generate any type of cave (i.e. complex labyrinths, simple linear caves or lairs of wild animals). The figure 6.9 shows a collection of caves formed by simple underground galleries and the figure 6.1 shows a more complicated cave. Unfortunately, it is difficult to perceive the playable differences only with static images.

Finally, the dungeon generation process is based on connectivity graphs, which determine the dungeon structure. Those graphs are defined by the designer, allowing the generation of any type of dungeon. We can observe some examples of the ampliability of the dungeon generator in the figure 6.20 (a dungeon similar to a traditional building formed by only one zone), 6.25 (a simple dungeon) and 6.31 (complex dungeons).

Additionally, the volumetric representation method improves the expandability aspect of the generator, because it can represent any type of construction.

Due to all of this, we can guess that the proposed method meets the expandability requirement.

Scalability

Requirement: 1. The method generates virtual worlds using an arbitrary scale.

On the one hand, the terrain scale is defined by a single input parameter (see section 5.3.2) and it affects the level of detail of the result (as shown in section 5.4.2, it is used as the frequency of the Perlin noise when the algorithm generates the heights and the material flow of the layers).

On the other hand, as in the terrain generator, the scale of the cave systems is also defined by one parameter (see section 6.2.3.3). As shown in section 6.2.4.1, this scale affects to the number of seeds of the Voronoi diagram and in the size of the underground cameras. That way, the level of detail of each underground camera depends on the scale because different underground camera sizes mean different number of voxels for each camera.

Finally, due to the special features of the dungeons, the dungeon generator is more limited in this aspect. In spite of that, as shown in section 6.3.4, the space where the dungeon is constructed is formed by a relative regular grid, so, the designer can assign an arbitrary lenght value to the lenght of each unit of the grid. That way, the scale of the dungeon can be modified.

Additionally, as shown in the chapter 4, the volume representation system requires less volumetric elements than traditional voxelised systems (table 4.1 shows this fact). So, the proposed representation system can store more data than traditional methods, helping in the representation of terrains with big scales.

Due to all of this, we can guess that the proposed method meets the scalability requirement.

Structure

> **Requirement: 2**. The terrains present recognizable structures coupled between them.

The factor which improves the level of structure of the results of the terrain generation methods are:

- The terrains are structured in recognizable layers and veins, similar to the real sedimentary grounds. Figure 5.13 shows some terrains generated with our method and compares them against real terrains.
- As shown in section 5.4.2, the materials of each layer are chosen using the affinities of the materials of the previous layer and the already selected materials of this layer, keeping the coherence among the terrain.
- The heights of each layer are influenced by the height of the previous layer, adjusting one to the other and coupling them (see section 5.4.2.1). Moreover, this similarity is calculated using the layer materials, which are defined by the designer. So, he can determine which materials are more malleable. We can see some examples of this fact in figure 5.12 and subfigures 5.13(d), 5.13(e), 5.13(f) y 5.13(g).

On another front, the cave generator gets the level of structure of their results with those factors:

- The visual structure is reached by the generation of connected underground galleries, integrating the whole cave system in a unique structure. We can see some examples in the subfigure 6.3(d) (in a two dimensional terrain) and in the figure 6.9 (in a three dimensional terrain).
- In order to reach the playable structure, the generation methods connect the different playable points of the POI using the affinities defined by the designer. This technique generates different playable structures (i.e. having high possibilities for generating a treasure next to an important encounter with an enemy). Figure 6.3(d) shows an example of cave with the POI roles assigned.

Finally, the dungeon generator is the method which generates the results with the highest level of structure of this research. It gets the structure of the results with the following factors:

- The method gets the playable structure generating the dungeon from graphs which define zones and rooms. Those elements are logical playable units with their own objective, allowing the multi-objective playability (for example, a global objective —rescue the princess at the end of the dungeon—, an objective of the zone —found the hidden door to the next zone—, and an objective of the room —beat an enemy—). Figures 6.26 and 6.31 show some examples of this structure.
- In the same way, the visual structure is given by the input graphs, structuring the dungeon in zones and rooms connected by corridors, door and stairs. As in the playable structure, we can see some examples in the figures 6.26 and 6.31.

Additionally, as in the previous requirements, the volumetric data representation method is able to store any type of structure, so, it does not limit this aspect of the results (see chapter 4).

All of this show that, apparently, the results are formed by recognizable structures coupled between them, so, we can guess that the proposed method meets the structure requirement.

Interest

> **Requirement: 2**. The results present explorable structures. Additionally, it motivates this exploration in any way.

First, due to the infiniteness of the terrains, they are explorable (see section 5.4.4). Moreover, the exploration is motivated by the distribution of mineral veins as playable resources. Figure 5.12 shows some terrains with mineral veins.

Second, the caves are also explorable by definition, and the exploration is motivated with the playability defined by the POIs of the cave (for example, giving a reward before killing an enemy, or with hidden treasures). Figure 6.1 shows a three dimensional cave suitable to be explored, and figure 6.3(d) shows the POIs of a cave and how they encourage the players to explore for searching the rewards.

Finally, the dungeon generator enables the designer to establish a strict playability. That way, the level of interest of the resulting dungeons is similar to the level of interest of the dungeons designed and modelled by the traditional way. Figures 6.26 and 6.31 show some examples where we can see how the playability is structured in layers: the

main objective is to reach the treasure room, but there are local objectives (find the keys to open the closed doors or beat the enemies). Moreover, the capability to create dungeon families allows virtual worlds to have many dungeons of the same type having a high level of interest.

Additionally, due to the capabilities of the representation method to store any structure (see chapter 4), it does not limit the possible interest of the results.

In conclusion, we can guess that the interest of the results achieves the required level. In spite of that, the interest of the virtual world is strongly dependent of factors such as the objective of the world or the ability of the designer to conceive interesting worlds and playable structures.

Speed

> **Requirement: 1**. The generation is performed in execution time, but previously to the interaction. Hybrid method between offline and online execution.

As shown in section 5.4.4, the terrain generation can be executed online. Additionally, we can see the time required by the process in the table 5.3.

On another front, the generation of the caves is a hybrid process (see section 6.2). In spite of that, there are several reasons to guess that it is possible to execute this process online:

- On the one hand, the steps of the process (see section 6.2.4.1), the randomness of the structure and the indefinite shape of the cave (see section 6.1.1) suggest that the cave systems are suitable to be constructed online.

- On the other hand, as shown in section 6.2.5.1, the step which requires more time to be executed is the generation of the Voronoi diagram and the Delaunay triangulation (see table 6.7). So, if the method generates those diagrams with a hybrid execution, it is possible to generate the rest of the cave online.

Finally, due to the restrictive features of the dungeons, the dungeon generator must be always executed in a hybrid way (see section 6.1.2).

These achievements show that the method meets the speed requirement.

Realism

Requirement: 2. The results are based in our reality, but they are not a simulation.

Even though any of the proposed methods are simulations of natural phenomenons, they are based in real elements to generate the results.

First, the factors which improves the level of realism of the results of the terrain generation methods are:

- As shown in figure 5.12, the terrains are based on layers of materials, imitating sedimentary grounds. We can see some examples in the section 5.5.3.
- The layers of the terrain are created with a Perlin noise generator (see section 5.2.1), which has been traditionally used to simulate natural elements [MKM89, SdKT+09]. We can see some examples in the figures 5.12 and 5.5.3. Moreover, this random generator is also used to create the underground material flow (see figure 5.3).
- As shown in section 5.4.2, the layers and veins of the terrain are formed by variable mixes of materials obtained by a probabilistic method based on the affinities of the nearby materials. For example, figure 5.12(b) shows a terrain in which the yellow layer is formed mixing two materials (there is no a yellow material defined in the input data). This fact contributes to increase the natural aspect of the terrains because, as usually happens, real terrains are formed by material mixtures too.
- The transition between neighbour layers is progressive, avoiding sharp ridges and giving a natural appearance. Moreover, the designer can specify the suitability of each material to be mixed with others (for example, a layer of sand can be mixed better than a layer of rock). We can see some examples of this fact in the figures 5.12 and 5.13.

Second, the level of realism of the caves obtained with our method is given by the following elements:

- As shown in section 6.2, cave generation includes the cave ramification, obtaining complex underground gallery systems. We can see an example of caves with ramifications in the subfigures 6.3(c) and 6.3(d), and a complex three-dimensional cave in the figure 6.1.

- All the underground galleries and cavities of the caves are created in cells of the Voronoi diagram. So, due to the physical irregularities of those cells (see subfigure 6.2(a)), underground cavities have a natural appearance.
- The generation of the underground galleries is performed keeping in mind the sinuosity of the cave (see section 6.2.4.4), avoiding an artificial appearance. We can see some examples in the figures 6.1 and 6.9.
- The playable structure of the caves organizes the playable events by the affinities defined by the designer, so, he determines the realism of the result. That is, if the designer wants a cave to represent the hideout of a group of thieves, he can define a POI to be the guard post at the entry of the cave or a POI to be the treasure chamber at the depth of the system. Subfigure 6.3(d) shows an example of a cave with the POI assigned.

Third, the factors which improves the level of realism of the dungeons created by our dungeon generator are:

- The dungeon generation tries to simulate real handmade structures such as mines or underground cities. For this purpose, as shown in the figure 6.31, it structures the results with groups of interconnected rooms, related between them by corridors, doors and stairs.
- As in the cave generation, the realism of the playable structure of the dungeons is completely determined by the designer distributing the events over the rooms. We can see some dungeons with the events distributed in the figures 6.26 and 6.31.

Finally, the proposed representation system requires less memory than traditional systems, so it can store more detailed terrains (see section 4.3) enhancing the realism. Additionally, due to its capabilities to store any type of structure, it does not limit the realism of the results.

Due to all of this, we can guess that the proposed method meets the realism requirement.

8.2 Comparison against other procedural generators

This section compares the proposed method against other procedural generators of virtual worlds using the key aspects and the metrics defined in the chapter 3.

Any of the compared generators uses an evaluation method which determines the level of each critical aspect objectively. Thus, to obtains the value for every aspect of each generator, we follow those steps:

1. If the scientific publication where the method is proposed explicitly mentions the aspect we want to evaluate, we assign the equivalent value of our metric. If the publication does not explicitly mention the aspect, the process continues with the next step.

2. We search in the scientific publication where the method is proposed any implicit reference to the aspect we want to evaluate, we analyse the method (as in the section section 8.1), and we assign a value of our metric to this aspect. If there is not implicit mention, we leave this aspect without an evaluation.

Additionally, if the method we are evaluating is an industrial method and has no scientific publication, the process is limited to the analysis performed for our method in section 8.1.

This process allows the assignment of a value of our metrics to every aspect of each generator[1], but, as we have seen in section 8.1, this value is only an estimation. In spite of that, we use these values to make a first approach to the evaluation and to the comparison of the generator.

We compare our method against those methods of the state of the art with a similar objective as ours. To this purpose, we have divided the comparison in three parts: terrain generation, cave generation and dungeon generation.

The comparison of the terrain generators has been made against fractal (see section 2.2.1.1) and evolutive generators (see section 2.2.1.2), but we have avoided the method proposed by Walsh and Gade [WG10] because their objective is to obtain beautiful images of a terrain and not the terrain generation itself. Additionally, we have added the generators of the most important video games with procedural volumetric virtual worlds (*Minecraft* [Moj11], *Terraria* [Re-11] and *Starbound* [Chu13]).

For the cave generation comparison, all the state of the art generators have been selected (see section 2.2.2). Additionally, as in the terrain comparison, we have added the generators of the most important video games with procedural volumetric virtual worlds (*Minecraft* [Moj11], *Terraria* [Re-11] and *Starbound* [Chu13]).

[1] If the value is impossible to assign, the table will show a hyphen(-).

	Innovation	Usability	Control	Expandability	Scalability
Our method	2	1	2	2	1
Minecraft	1	-	-	-	0
Terraria	1	-	-	-	0
Starbound	2	-	-	-	0
Fractal terrains	1	0	1	0	1
Ong *et al.*	1	1	2	2	1
Ashlock *et al.*	2	1	2	2	1
Frade *et al.*	1	0	3	0	1
Togelius *et al.*	1	0	2	0	0
Raffe *et al.*	0	1	2	2	1

Tabla 8.1: Comparison against terrain generators (designer aspects).

Finally, the dungeon generation comparison has been made against Dormans [Dor10] and Sorensons and Pasquier [SP10] methods. Those methods are the unique methods of the state of the art which generate playable dungeons. Additionally, as in the previous two parts, we have added the generators of the most important video games with procedural volumetric virtual worlds (*Minecraft* [Moj11], *Terraria* [Re-11] and *Starbound* [Chu13]).

We can see the comparison against other terrain generators at the tables 8.1 and 8.2, against other cave generators at the tables 8.3 and 8.4, and against other dungeon generators at the tables 8.5 and 8.6.

In spite of some generators are better than ours in some aspects, we can observe that there is no other method that meets all the requirements proposed in the section 3.2.3.

	Structure	Interest	Speed	Realism
Our method	2	2	1	2
Minecraft	1	2	2	1
Terraria	2	2	1	1
Starbound	2	2	1	1
Fractal terrains	2	0	2	2
Ong *et al.*	1	1	1	-
Ashlock *et al.*	1	1	1	-
Frade *et al.*	2	1	1	-
Togelius *et al.*	1	2	1	-
Raffe *et al.*	1	2	0	-

Tabla 8.2: Comparison against terrain generators (user aspects).

	Innovation	Usability	Control	Expandability	Scalability
Our method	2	1	2	2	1
Minecraft	2	-	-	-	0
Terraria	2	-	-	-	0
Starbound	2	-	-	-	0
Boggus y 0 Crawfis	1	-	-	0	1
Peytavie *et al.*	2	0	3	2	1
Johnson *et al.*	2	0	1	0	1
Juncheng *et al.*	1	0	2	0	1

Tabla 8.3: Comparison against cave generators (designer aspects).

	Structure	Interest	Speed	Realism
Our method	2	2	1	2
Minecraft	2	1	2	2
Terraria	2	2	1	2
Starbound	2	1	1	2
Boggus y Crawfis	2	1	-	3
Peytavie *et al.*	2	1	0	3
Johnson *et al.*	0	1	2	1
Juncheng *et al.*	2	1	-	2

Tabla 8.4: Comparison against cave generators (user aspects).

	Innovation	Usability	Control	Expandability	Scalability
Our method	2	1	2	2	1
Minecraft	1	-	-	-	0
Terraria	1	-	-	-	0
Starbound	1	-	-	-	0
Dormans	1	0	2	2	-
Sorensons y Pasquier	1	1	1	2	-

Tabla 8.5: Comparison against dungeon generators (designer aspects).

	Structure	Interest	Speed	Realism
Our method	2	2	1	2
Minecraft	2	0	2	2
Terraria	2	1	1	2
Starbound	2	1	1	2
Dormans	2	2	-	-
Sorenson y Pasquier	1	2	0	-

Tabla 8.6: Comparison against dungeon generators (user aspects).

8.3 Weaknesses and strengths of the proposed method

This section describes the weaknesses and strengths of the proposed generator.

Weaknesses

We can observe the following weak points:

- Terrain generator is based on sedimentary terrains, being impossible the generation of other type of grounds.
- Resulting terrains are formed by layers generated by modified Perlin noise, giving to their surface a simple aspect. This problem could be tackled using any solution to simulate the erosion over existing generated terrains (see section 2.2.1.1).
- As we have seen before, the cave generation requires a lot of time. In spite of this, it is possible to perform the process in two steps, softening the effect of this problem.
- Resulting cave systems have a high variability in the cave configuration (number of POIs, number of galleries...) and in the time required to perform the generation. This problem could be avoided defining the input data with care.
- If the graph which defines a dungeon is too complex, the generation could be impossible. For example, if a graph defines a zone with a lot of interconnected rooms, the solution may not exist.

Strengths

On the other hand, we can observe the following strengths:

- In order to evaluate our method against the requirements, it is necessary to implement the method in an interactive application and perform experiments with groups of designers and users, but, as shown in section 8.1, there are a many circumstantial evidences to guess that the method meets the requirements. Due to this fact, we are able to say that our method is valid to be directly used in the industry.

- The customization level enables the method to be used for many objectives, such as military simulations, role playing games or adventure games. In conclusion, the method is valid for any application which requires big volumetric terrains.
- The resource distribution over the terrain using mineral veins fits perfectly with existing video games which base their playability on the survival experience and the resources gathering.
- The cave generator allows any type of playability, defined by the designer using the POIs. This approach fits perfectly with all the commercial video games based on the exploration.
- The dungeon generator enables the designer to define the dungeon with a very high level of detail. Obtained dungeons are much more complex than the dungeons of the most popular games with procedural volumetric worlds (i.e. *Minecraft* [Moj11], *Terraria* [Re-11], o *Starbound* [Chu13]), and they are similar to the dungeons presented in no procedural games (i.e. *Skyrim* 1.1(c)).

8.4 Validation of the initial hypothesis

Returning to the initial hypothesis of this research, we can check that it is formed by two different parts:

> «*It is possible to generate volumetric virtual worlds achieving all the explicit and implicit requirements proposed by the community, using no supervised methods based on spacial techniques and artificial intelligence. Aditionally, it is posible to synthesise those requirements and the metrics to evaluate them.*»

First, we have identified and synthesised the key aspects for procedural generators. This process has been successfully performed, collecting the explicit and implicit key aspects from the literature and the industry, and proposing a set of metrics to evaluate them. Additionally, we have analysed the most used generators, proposing the minimum level of the metric of every aspect a procedural generator should meet. To the best of our knowledge, this is the first research which advances the standardization of the methods and the criteria to evaluate procedural generators.

Second, we have proposed a new volumetric virtual world generator using the previously proposed requirements as a guide. Unfortunately, for the validation of those

requirements additional experiments are needed implementing the method in a real video game and working with real designers and users. In spite of that, we have collected many circumstantial evidences which enable us to guess that the method meets all the requirements.

We can state that we have accomplished the first four operational objectives (the identification of the key aspects of the literature and the industry, the requirements proposal and the generator design), but in the case of the fifth operational objective (the method evaluation) we have only made a first approach. So, while we have achieved the first specific objective, we have only met the second one partially.

Due to all of this, we can state that this research has reached a partial success in the validation of the initial hypothesis, and it leaves the complete validation for future work.

8.5 Contributions

The main contributions of this research are:

- We have collected a set of critical aspects for procedural generators. Additionally, we have proposed a set of metrics to evaluate them.
- We have proposed a set of requirements for procedural generators using the key aspects and their metrics.
- We have proposed a volumetric data representation system which requires less space than traditional representations based on voxels.
- We have proposed a procedural method to generate virtual worlds, including the volumetric terrain and the playable layer (formed by the caves and the dungeons). This generator has been designed taking into account the previously defined requirements.
- We have proposed a novel method for volumetric terrain generation based on sedimentary grounds. All the results and aspects of this generator show that it meets the proposed requirements.
- Additionally, we have proposed a new method for cave system generation based on Voronoi diagrams. This method enables the designer to define the playable points which are going to be in the cave and how they are related. As in the

terrain generator, all the results and aspects of this method show that it meets the proposed requirements.

- Finally, we have also proposed a new method for dungeon generation based on building layout generation methods using pressure techniques. This method is able to create families of dungeons with a controlled variability and an strict playability defined by the designer. As in the two previous generators, all the results and aspects of this method show that it meets the proposed requirements.

Thus, we have proposed a new method to generate virtual worlds and we have take the first step to the organization and standardization of the criteria to evaluate this type of generators.

8.6 Future work

This section proposes three lines for future work to continue with this research.

Final evaluation

As shown in the section 8.1 the evaluation of the method requires additional experiments which are out of the scope of this dissertation, so, the main line of work to continue this research is this evaluation.

On the one hand, in order to evaluate the designer oriented aspects (see section 3.1.1) it is necessary to perform sessions with designers using the proposed method, and evaluate it with a survey.

On the other hand, in order to evaluate the user oriented aspects (see section 3.1.2) we propose the implementation of the method in a real video game, evaluating it with interactive sessions and surveys among users.

Higher level of detail

The method proposed in this research generates complete virtual worlds, but it does not delve into the detail of each element. Due to this fact, we propose a future line of work to improve this aspect.

Some suitable aspects to focus on are:

- The surface of the terrain is simple and does not present elements such as rivers o peaks. For this purpose, we propose the research of a possible application of any of the erosion simulation methods presented in the section 2.2.1. One of the biggest challenges of this line of work is the development of a method which may be executed in real time.
- The caves obtained with our method present a natural appearance due to the irregular shape of the Voronoi diagram cells. In spite of that, the realism of the galleries is limited by the lack of elements such as stalactites and stalagmites. We propose a new line of work to search for a method to generate this type of elements.

Improvement of the generation speed

As shown in previous chapters, the method can be executed in a hybrid way (value **1** in the metric of the speed aspect). Additionally, some of the steps are really suitable to improve their execution speed (for example, the Voronoi diagram construction in the cave generator). We think that it is possible to achieve real time execution of the generator improving the execution time of those steps.

8.7 Final remarks

Due to the rapid growth of the computer graphics industry (video games, simulators, special effects, etc.), it is impossible to obtain a definitive method to generate procedural virtual worlds. The gradual increase of the required detail implies that procedural generators must be continuously evolving. Additionally, due to the special characteristics of the video games with procedural content, procedural generators are widely used in many video game genres, increasing the complexity of the problem.

Due to this situation, the research of the new methods and techniques to procedurally generate virtual worlds and other type of content needs to grow in parallel with the technical and social evolution of the graphic interactive applications.

Apéndices

«La magia es impresionante, pero ahora Minsc está al mando. ¡Espadas para todos!»

Minsc (*Baldur´s Gate*, BioWare, 1998)

CAPÍTULO

A

Desarrollo del prototipo *Worldfoundry*

En este apéndice mostramos el prototipo utilizado en esta investigación para llevar a cabo todos los experimentos que la conforman. Este prototipo, al que hemos llamado *Worldfoundry*, es una *suite* de generación de mundos virtuales procedurales completa a la que se pueden agregar diferentes herramientas, entre las cuales se encuentran todos los algoritmos propuestos en esta tesis.

Worldfoundry es una herramienta *software* compuesta por varios módulos dedicados a diferentes tareas, que se ha desarrollado utilizando la tecnología *Microsoft.NET*[1], en concreto el lenguaje de programación C#[2], y empleando *Windows Forms* para la interfaz gráfica de usuario y *Microsoft DirectX 11*[3] para el desarrollo de la tecnología gráfica. Como enlace entre la API *Microsoft DirectX 11* y la plataforma *Microsoft.NET* se utiliza la librería *SlimDX*[4].

Lo que resta de capítulo se encuentra estructurado de la siguiente manera: en la

[1] http://www.microsoft.com/net

[2] http://www.ecma-international.org/publications/standards/Ecma-334.htm

[3] *Software Development Kit* (SDK) disponible en http://www.microsoft.com/en-us/download/details.aspx?id=6812

[4] *Software Development Kit* (SDK) disponible en http://slimdx.org/

sección A.1 describimos la estructura interna de *Worldfoundry* y enumeramos sus módulos definiendo las relaciones entre ellos; y en las secciones A.2, A.3 y A.4 describimos en profundidad cada componente del prototipo.

A.1 Módulos desarrollados

Worldfoundry está compuesto por tres módulos principales cuya relación se representa en la figura A.1.

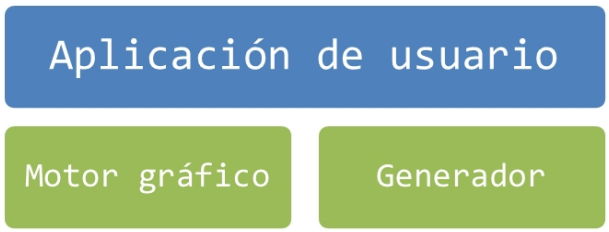


Figura A.1: Estructura de *Worldfoundry*.

Por un lado, el **motor gráfico** es el encargado de gestionar todo lo referente a la visualización de la escena, la gestión de memoria para los recursos gráficos y los *shaders* que se ejecutan en la tarjeta gráfica.

Por otro lado, el **generador** es la librería encargada de crear el mundo virtual. En este módulo se encuentran los algoritmos de generación procedural propuestos en esta tesis.

Por último, existe una **aplicación de usuario** que hace uso de los dos módulos anteriores para permitir al diseñador utilizar la herramienta mediante una cómoda interfaz gráfica.

A.2 Motor gráfico

El motor gráfico utilizado por *Worldfoundry* lo hemos desarrollado expresamente para este propósito, por lo que sus componentes son los estrictamente necesarios para este objetivo. A pesar de ello, lo hemos diseñado de forma modular, de tal manera que es posible añadir nuevos elementos y características al motor con gran facilidad, y también mejorarlo hasta obtener un motor gráfico convencional completo.

En este momento, el motor se divide en cuatro componentes principales: escena, encargada de gestionar todos los parámetros de la escena y sus nodos; *render*, encargado

de dibujar la escena en el lienzo; *input*, encargado de leer y actualizar el estado de los dispositivos de entrada; y recursos, encargado de gestionar el uso de memoria de los recursos gráficos.

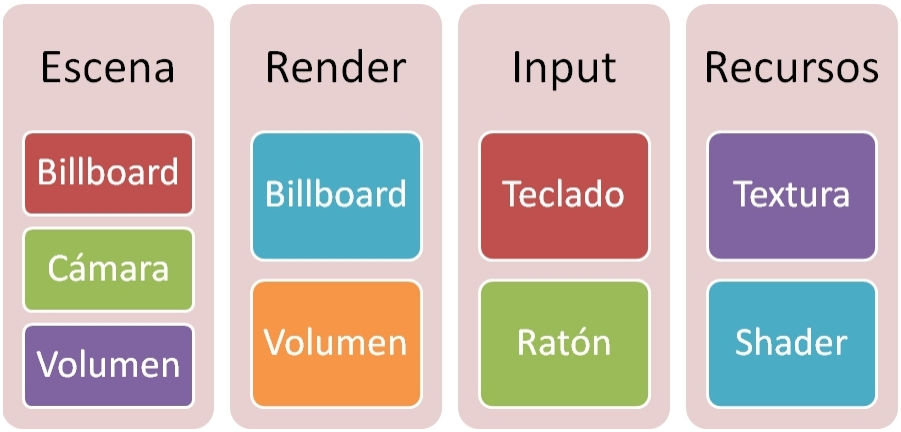


Figura A.2: Estructura del motor gráfico.

Además, como podemos ver en la figura A.2, cada uno de estos componentes contiene una serie de elementos relacionados con su tarea.

A.2.1 **Escena**

Este módulo se encarga de gestionar el estado de la escena en cada momento. Sus tareas principales son actualizar los diferentes nodos que la componen y gestionar la jerarquía entre ellos.

Actualmente existen solamente tres tipos diferentes de nodos (seis si incluimos los tres tipos diferentes de cámaras), aunque crear nuevos tipos (por ejemplo, mapas de alturas o sistemas de partículas) es un proceso sencillo.

A.2.1.1 ***Billboard***

Los nodos de tipo *billboard* son todos aquellos elementos de la escena que, aunque estén situados en un espacio tridimensional, se representan mediante una imagen bidimensional orientada siempre hacia el punto de vista de la cámara.

Hace unos años, cuando el número de polígonos que una tarjeta gráfica era capaz de manejar estaba enormemente limitado, se utilizaban *billboards* para representar múltiples elementos gráficos tales como árboles o personajes. Hoy la utilización de este

tipo de elementos se centra en la representación de poblaciones muy grandes de objetos (por ejemplo, las hojas de los árboles) o de marcadores en la escena tridimensional (por ejemplo, un indicador del siguiente punto del mapa a donde el jugador debe dirigirse).

En *Worldfoundry* los *billboards* se utilizan como marcadores de los puntos de interés para depurar el algoritmo de generación de cuevas (ver capítulo 6).

A.2.1.2 **Cámara**

Como su propio nombre indica los nodos de tipo cámara son aquellos puntos desde los cuales se visualiza la escena. Aunque es posible utilizar múltiples cámaras, el motor gráfico debe tener siempre una única cámara activa, que es desde donde el componente *render* dibuja la escena.

El motor tiene tres tipos diferentes de cámaras:

- **Cámara en primera persona:** Cámara que se encuentra exactamente detrás de los ojos del observador. Este tipo de cámaras se utilizan en todas aquellas aplicaciones en las que controlamos a un personaje desde su propio punto de vista.
- **Cámara en tercera persona:** Cámara que se encuentra centrada en un elemento entorno al cual puede orbitar, acercarse o alejarse. Este tipo de cámaras se utilizan normalmente en aplicaciones gráficas donde es necesario inspeccionar un objeto desde múltiples ángulos. Además, también se emplean para visualizar aplicaciones interactivas donde el jugador maneja a un personaje desde un punto de vista externo.
- **Cámara ortogonal:** Cámara sin perspectiva, es decir, todos los elementos se dibujan a su tamaño original sin tener en cuenta la distancia a la que se encuentran. Este tipo de cámaras son muy utilizadas para aplicaciones que requieran vistas isométricas.

A.2.1.3 **Volumen**

Los nodos de tipo volumen representan a cualquier elemento que esté compuesto por datos volumétricos. En el prototipo que nos ocupa se utilizan para representar los terrenos volumétricos generados por los algoritmos propuestos.

Además, este tipo de nodo está diseñado para gestionar los datos volumétricos con el sistema de representación propuesto en el capítulo 4.

A.2.2 *Render*

El módulo *render* es el encargado de dibujar la escena desde el punto de vista de la cámara activa en cada momento. Para ello, utiliza la API gráfica *Microsoft DirectX* en su versión 11, conectándola a la plataforma *Microsoft.NET* con la librería *SlimDX*.

Además, este módulo se encarga de lanzar los diferentes *shaders* en el *hardware* gráfico. Estos *shaders* son pequeños trozos de código que se ejecutan en un entorno altamente paralelizado (el procesador gráfico). Existen dos tipos diferentes: *vertex shaders*, que se ejecutan una vez por cada vértice de la geometría que se está dibujando, y *pixel shaders*, que se ejecutan una vez por cada píxel en pantalla del elemento que se está dibujando.

En *Worldfoundry* se utilizan para múltiples propósitos, aunque el más significativo es la deformación de las texturas de los volúmenes dependiendo del flujo interno del terreno.

Este módulo está compuesto por los elementos *billboard* y volumen, que son los encargados de dibujar en el lienzo los nodos de escena con el mismo nombre.

A.2.3 *Input*

El módulo *input* es el encargado de recibir las órdenes del teclado y del ratón, y de actualizar el estado de la escena dependiendo de ellas. Al igual que el módulo *render* está basado en *Microsoft DirectX 11*, aunque en este caso se utiliza solamente su componente *Direct Input*, encargado de gestionar los dispositivos de entrada.

Worldfoundry utiliza este módulo para rotar, acercar y alejar la cámara alrededor del terreno, ya que el resto de entradas de datos se realiza mediante formularios *Windows Forms*.

A.2.4 **Recursos**

Este módulo tiene como cometido la gestión de la memoria que ocupan los diferentes recursos gráficos utilizados por la escena. Además, también incluye las rutinas necesarias para cargar de disco dichos elementos.

Una de las principales características de este módulo es la de servir como una capa adicional entre el disco y la aplicación, impidiendo entre otras cosas, que un mal uso de las rutinas de lectura cargue en memoria varias veces el mismo elemento.

Como podemos ver en la figura A.2, está compuesto por dos elementos. Textura es el encargado de representar en memoria cualquier tipo de imagen, mientas que *shader* es el elemento que representa el código que después el módulo *render* ejecuta en el *hardware* gráfico. La carga de los dos elementos se realiza mediante funciones proveídas por *Microsoft DirectX*.

A.3 Motor de generación

Este componente comprende el proceso completo de generación de mundos virtuales propuesto en esta tesis y está formado por los métodos de generación de terrenos volumétricos (ver capítulo 5), de sistemas de cuevas y de mazmorras (ver capítulo 6).

Al igual que el motor gráfico, el motor de generación incluye exclusivamente los métodos de generación necesarios para los experimentos llevados a cabo en esta tesis, pero debido a la manera en la que se encuentra diseñado y construido es posible aumentar su capacidad fácilmente; de manera que este motor además de ser una herramienta para esta investigación, puede convertirse en un *framework* de generación procedural completo.

A.4 Aplicación de usuario

La herramienta *Worldfoundry* une los dos módulos anteriores mediante una aplicación de usuario que incluye todo lo necesario para facilitar el uso de los algoritmos de generación. La figura A.3 muestra cómo se divide esta aplicación en componentes y cuáles son los elementos que incluyen dichos componentes.

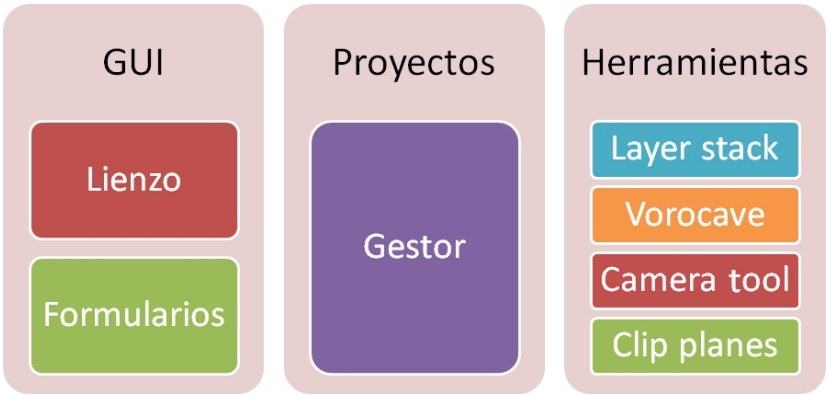


Figura A.3: Estructura de la aplicación de usuario.

A.4.1 GUI

El módulo GUI comprende todas las interfaces de usuario del prototipo, a excepción de los formularios específicos de cada herramienta, que se incluyen en el respectivo módulo de cada herramienta. Además, este módulo incluye un elemento lienzo, que permite mostrar la escena tridimensional renderizada por el motor gráfico en un panel de la ventana principal.

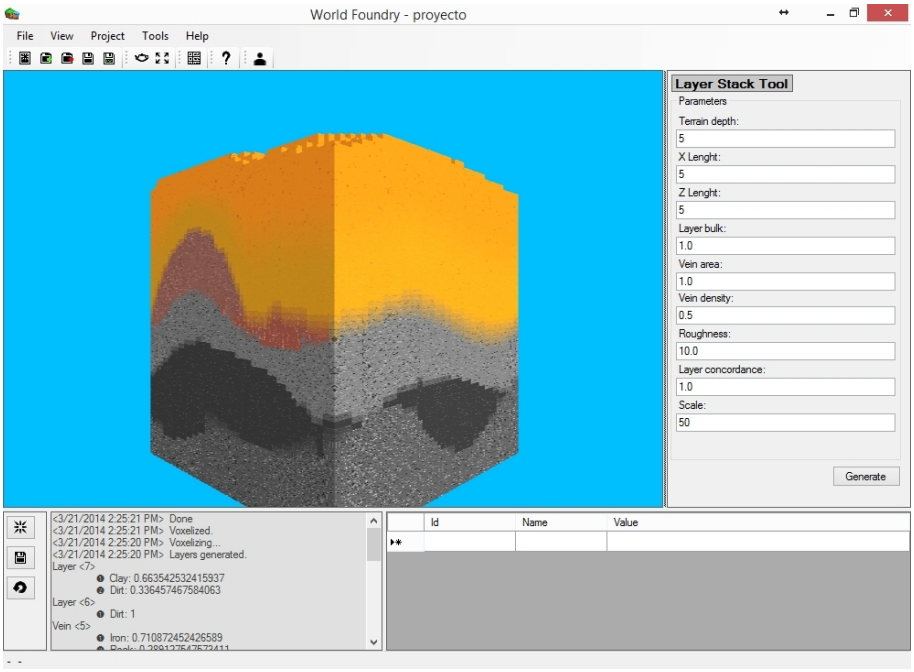


Figura A.4: Ventana principal de la aplicación.

La ventana principal está formada por los siguientes componentes:

- **Menú principal:** Medio para acceder a todas las opciones de la aplicación. Permite gestionar los proyectos y mostrar u ocultar las diferentes herramientas.
- **Barra de accesos directos:** Accesos directos a las funcionas más utilizadas de la aplicación.
- **Lienzo de escena:** Lienzo sobre el que se dibuja la escena. Este componente se encuentra desarrollado en el módulo que contiene el motor gráfico.
- **Paneles de herramientas:** Contienen las interfaces gráficas de las herramientas y su contenido es dinámico, por lo que pueden mostrar los controles de cual-

quier herramienta. Pueden desplegarse dos paneles diferentes, uno a cada lado del lienzo de escena.

- **Consola:** Consola donde se muestran diferentes mensajes sobre el estado interno de la aplicación. Tiene asociados una serie de botones para crear *logs*, limpiar la ventana o ir a la posición del cursor.

- **Rejilla de datos:** Tabla en la cual puede visualizarse información de la escena, como los nodos de los que se compone o los recursos cargados en memoria en cada momento.

A.4.2 Gestión de proyectos

Worldfoundry basa su funcionamiento en proyectos que representan un único terreno volumétrico. Cada proyecto se almacena en un directorio con los siguientes ficheros:

- **<Proyecto>.wofo** (***Wo****rld****fo****undry*): Archivo principal del proyecto. Recoge información acerca de la fecha de creación y sus modificaciones, así como de las rutas al resto de archivos que componen el proyecto.

- **<Elementos>.woel** (***Wo****rldfoundry* ***El****ements*): Contiene los datos de entrada del método de generación de terrenos (ver capítulo 5). En él se definen los elementos que componen el terreno.

- **<Terreno>.woft** (***Wo****rld****f****oundry* ***T****errain*): Cabecera del terreno. Contiene información sobre el tamaño del *chunk* utilizado por el terreno (ver capítulo 4) y un listado sobre los *chunks* ya creados y almacenados en disco.

- **<Cueva>.woca** (***Wo****rldfoundry* ***Ca****ve*): Fichero que contiene los datos de entrada del método de generación de cuevas (ver sección 6.2). Este fichero es opcional y solamente se requiere en el momento en el que se desea lanzar este algoritmo.

- **<Mazmorra>.wsch** (***W****orldfoundry* ***Sch****maltzberg*): Fichero con las definiciones de las mazmorras (ver capítulo 6.3). Este fichero es opcional y solamente se requiere en el momento en el que se desea lanzar este algoritmo.

- **Carpeta con los ficheros del terreno:** Carpeta que contiene todos los ficheros .wchk (***W****orldfounfry* ***Ch****un****k***) que componen el terreno. Cada uno de estos ficheros almacena la información de un *chunk*.

A.4.3 Herramientas

Las herramientas son el verdadero corazón del funcionamiento de *Worldfoundry*, ya que permiten realizar diferentes acciones sobre el proyecto o sobre la manera de visualizar la escena. Cada una de ellas tiene su propia interfaz personalizada, y pueden cargarse y descargarse en la ventana principal personalizando la interfaz general del programa para adecuarla a diferentes propósitos.

Las herramientas desarrolladas son las siguientes:

A.4.3.1 Herramienta *Layer stack*

Esta herramienta es la encargada de lanzar el algoritmo de generación de terrenos expuesto en el capítulo 5. Utiliza el archivo .woel del proyecto para leer las características de los materiales y sus relaciones, mientras que los parámetros generales los obtiene directamente de su interfaz (como podemos ver en la figura A.5).

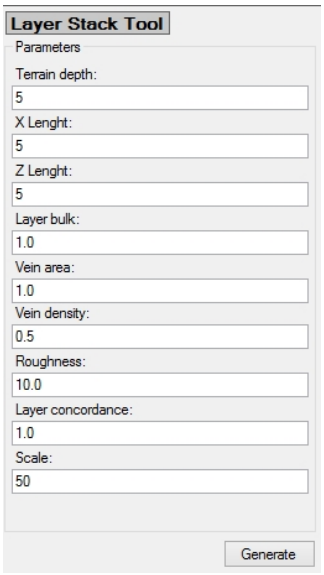


Figura A.5: Interfaz de la herramienta *Layer stack*.

Además, esta interfaz pide al usuario el tamaño del terreno en los ejes de coordenadas x y z, ya que para este prototipo solamente se ha implementado la generación de pequeñas porciones del terreno.

A.4.3.2 **Herramienta *Vorocave***

La herramienta *Vorocave* es la encargada de ejecutar el algoritmo de generación de cuevas expuesta en el capítulo 6. Para que esta herramienta esté disponible es necesario un proyecto que ya tenga un terreno generado (ya sea por la herramienta *Layer stack* o manualmente).

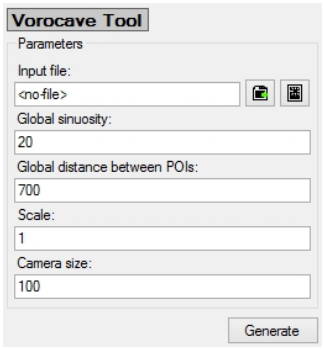


Figura A.6: Interfaz de la herramienta *Vorocave*.

En su interfaz (que podemos ver en la figura A.6) se puede seleccionar el fichero .woca con las definiciones de los puntos de interés (si se deja en blanco la aplicación busca este archivo en la carpeta del proyecto) e introducir los valores de los parámetros generales requeridos por el método. Podemos apreciar que no es posible introducir la profundidad del terreno, debido a que dicho valor se obtiene directamente del terreno sobre el que vamos a ejecutar este método.

A.4.3.3 **Herramienta *Schmaltzberg***

Esta herramienta se encarga de ejecutar el algoritmo de generación de mazmorras (ver capítulo 6). Al igual que la herramienta *Vorocave*, para que esté disponible es necesario un proyecto que ya tenga un terreno generado.

La figura A.7 muestra su interfaz; en ella se puede seleccionar el fichero donde se establecen los grafos que definen la mazmorra e introducir los valores para los parámetros generales requeridos por el método.

A.4.3.4 **Herramienta *Camera***

Esta herramienta se utiliza para modificar el tipo de cámara activa y sus parámetros, a través de la cual se está visualizando la escena.

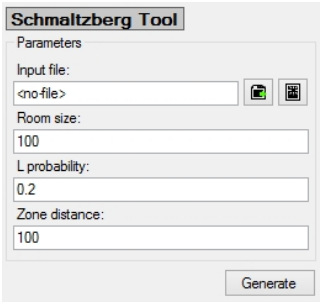


Figura A.7: Interfaz de la herramienta *Schmaltzberg*.

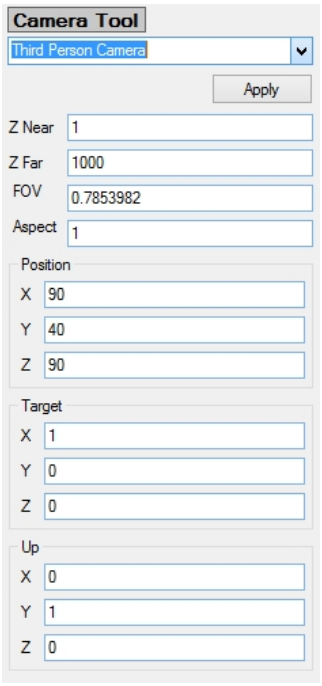


Figura A.8: Interfaz de la herramienta *Camera*.

Como vemos en la figura A.8, la interfaz de esta herramienta contiene un panel desplegable en el que se selecciona el tipo de cámara que queremos utilizar. Además, al modificar el valor seleccionado en este desplegable cambian los parámetros que se muestran debajo.

A.4.3.5 Herramienta *Clip planes*

Por último, la herramienta *Clip planes* nos permite establecer una serie de planos de corte en el terreno para poder ver su interior. Estos planos se definen perpendiculares

a uno de los ejes con una profundidad determinada y se establece una dirección en el eje, así, se evita el dibujado de los *voxels* a ese lado del plano.

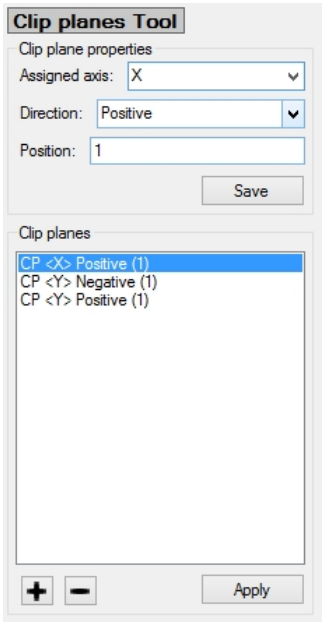


Figura A.9: Interfaz de la herramienta *Clip planes*.

En la figura A.9 podemos ver la interfaz de esta herramienta. Está compuesta por una lista, donde se enumeran los *clip planes* existentes, se agregan nuevos o se eliminan; y una serie de campos, donde se modifican los parámetros de cada uno de ellos.

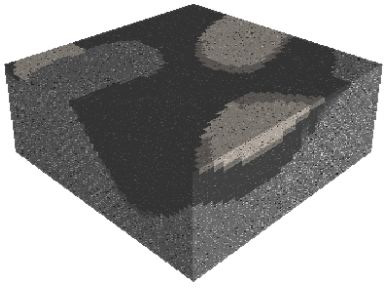


Figura A.10: Visualización de un terreno sobre la que se ha aplicado un *clip plane* perpendicular al eje y.

La figura A.10 muestra un terreno con un *clip plane* perpendicular al eje y que evita que se dibujen todos aquellos *voxels* que se encuentran a partir de dicho plano.

Bibliografía

Videojuegos

[Aso09] Asobo Studio. *FUEL*, junio 2009. http://www.codemasters.com/es/fuel/360/.

[Ata72] Atari Inc. *PONG*, 1972.

[Ava06] Avalanche Studios. *Just Cause*, septiembre 2006. http://www.justcause.com/.

[Bay06] Bay 12 Games. *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress*, agosto 2006. http://www.bay12games.com/dwarves/.

[BB84] Braben, D. y Bell, I. *Elite*, septiembre 1984.

[Bet96] Bethesda Softworks. *The Elder Scrolls II: Daggerfall*, agosto 1996. http://www.elderscrolls.com/daggerfall/.

[Bet11] Bethesda Game Studios. *The Elder Scrolls V: Skyrim*, noviembre 2011. http://www.elderscrolls.com/skyrim.

[Bli96] Blizzard North. *Diablo*, septiembre 1996.

[Cap96] Capcom. *Resident Evil*, marzo 1996. http://www.residentevil.com/.

[Chu13] Chucklefish Games. *Starbound*, diciembre 2013. http://playstarbound.com/.

[Evo10] Evolutionary Games. *Galactic Arms Race*, 2010. http://galacticarmsrace.blogspot.com.es/.

[ID 91] ID Software. *Catacomb 3D*, noviembre 1991. http://www.idsoftware.com/games/vintage/catacomb.

[Max99] Maxis. *SimCity 3000*, enero 1999. http://www.simcity.com/es_ES/product/simcity3000.

[Max08] Maxis. *Spore*, septiembre 2008. http://www.spore.com/.

[Moj11] Mojang. *Minecraft*, noviembre 2011. https://minecraft.net/.

[MS05b] Mateas, M. y Stern, A. *Façade, a one-act interactive drama*, 2005. http://www.interactivestory.net.

[Nin85] Nintendo. *Super Mario Bros.*, septiembre 1985.

[Nin86] Nintendo. *The Legend of Zelda*, febrero 1986. http://www.zelda.com/.

[Nov92] NovaLogic. *Comanche Maximum Overkill*, 1992.

[Páz84] Pázhitnov, A. *Tetris*, junio 1984. http://www.tetris.com/.

[Re-11] Re-Logic. *Terraria*, mayo 2011. http://www.terraria.org/.

[.th04] .theprodukkt. *.kkrieger*, 2004.

[Wes90] Westwood Associates. *Dungeons & Dragons: Eye of the Beholder*, 1990.

Herramientas

[Ble95] Blender Fundation. *Blender*, 1995. http://www.onyxtree.com/.

[Bun03] Bundysoft. *L3DT*, 2003. http://www.bundysoft.com/.

[Esr08] Esri R&D Center Zurich. *CityEngine*, 2008. http://www.esri.com/software/cityengine.

[Gam] Gamr7. *Urban PAD*.

[Gre96] Greenworks. *Xfrog*, 1996. http://xfrog.com/.

[Int09] Interactive Data Visualization, Inc. *SpeedTree*, 2009. http://www.speedtree.com/.

[Ony92] Onyx Computing. *Onix*, 1992. http://www.onyxtree.com/.

[Pix] PixelActive. *CityScape*.

[Pla09] Planetside Software. *TerraGen 2*, 2009. http://planetside.co.uk/.

[Ros06] Rosenberg, J. *GeoControl*, 2006. http://www.geocontrol2.com/.

[Wor05] World Machine Software, LLC. *World Machine*, 2005. http://www.world-machine.com/.

Publicaciones

[AGB05] Ashlock, D. A., Gent, S. P., y Bryden, K. M. Evolution of l-systems for compact virtual landscape generation. *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC '05)*, páginas 2760–2767. IEEE, 2005.

[AMO12] Arce, V., Martín, M., y Ortega, J. A. *Global Entertainment and Media Outlook: 2012-2016*. Informe técnico, PricewaterhouseCoopers S.L., 2012.

[ASA07] Anh, N., Sourin, A., y Aswani, P. Physically based hydraulic erosion simulation on graphics processing unit. *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia (GRAPHITE '07)*, páginas 257–264. ACM, 2007.

[Ash10] Ashlock, D. A. Automatic generation of game elements via evolution. *Proceedings of the 6th International Conference on Computational Intelligence and Games (CIG '09)*, páginas 289–296. IEEE, 2010.

[Aur91] Aurenhammer, F. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3):páginas 345–405, 1991.

[BA05] Belhadj, F. y Audibert, P. Modeling landscapes with ridges and rivers: bottom up approach. *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '05)*, páginas 447–450. ACM, 2005.

[BC09a] Boggus, M. y Crawfis, R. Explicit generation of 3d models of solution caves for virtual environments. *Proceedings of the 2009 International Conference on Computer Graphics & Virtual Reality (CGVR '09)*, páginas 85–90. 2009.

[BC09b] Boggus, M. y Crawfis, R. Procedural creation of 3d solution cave models. *Proceedings of the 20th IASTED International Conference on Modelling and Simulation*, páginas 180–186. IASTED, 2009.

[BC10] Boggus, M. y Crawfis, R. Prismfields: a framework for interactive modeling of three dimensional caves. *Proceedings of the 6th International Conference on Advances in Visual Computing (ISVC '10)*, páginas 213–221. Springer, 2010.

[BE92] Bern, M. y Eppstein, D. Mesh generation and optimal triangulation. *Computing in Euclidean geometry*, 1(1):páginas 23–90, 1992.

[BF01] Benes, B. y Forsbach, R. Layered data representation for visual simulation of terrain erosion. *Proceedings of the 17th Spring Conference on Computer graphics (SCCG '01)*, páginas 80–86. IEEE, 2001.

[BHVW00] Bruls, M., Huizing, K., y Van Wijk, J. J. Squarified treemaps. *Proceedings of Joint Eurographics and IEEE TCVG Symposium. on Visualization (TCVG '00)*, páginas 33–42. IEEE, 2000.

[BMJ+11] Beneš, B., Massih, M. A., Jarvis, P., Aliaga, D. G., y Vanegas, C. A. Urban ecosystem design. *Proceedings of the 2011 Symposium on Interactive 3D Graphics and Games (i3D '11)*, páginas 167–174. ACM, 2011.

[BS95] Bukowski, R. W. y Séquin, C. H. Object associations: a simple and practical approach to virtual 3d manipulation. *Proceedings of the 1995 Symposium on Interactive 3D Graphics (i3D '95)*, páginas 131–ff. ACM, 1995.

[BŠMM11] Beneš, B., Štáva, O., Měch, R., y Miller, G. Guided procedural modeling. *Computer Graphics Forum*, 30(2):páginas 325–334, 2011.

[BV04] Boyd, S. y Vandenberghe, L. *Convex optimization*. Cambridge University Press, 2004.

[CCZ11a] Cui, J., Chow, Y. W., y Zhang, M. Procedural generation of 3d cave models with stalactites and stalagmites. *International Journal of Computer Science and Network Security*, 11(8):páginas 94–101, 2011.

[CCZ11b] Cui, J., Chow, Y. W., y Zhang, M. A voxel-based octree construction approach for procedural cave generation. *International Journal of Computer Science and Network Security*, 11(6):páginas 160–168, 2011.

[CD98] Chopard, B. y Droz, M. *Cellular automata modeling of physical systems*. Cambridge University Press, 1998.

[CEW+08] Chen, G., Esch, G., Wonka, P., Müller, P., y Zhang, E. Interactive procedural street modeling. *ACM Transactions on Graphics (TOG)*, 27(3), artículo No. 103, 2008.

[CM06] Compton, K. y Mateas, M. Procedural level design for platform games. *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE '06)*, páginas 109–111. AAAI, 2006.

[dCB09] Carpentier, G. J. de y Bidarra, R. Interactive gpu-based procedural heightfield brushes. *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG '09)*, páginas 55–62. ACM, 2009.

[Del34] Delaunay, B. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:páginas 793–800, 1934.

[DHL+98] Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., y Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems. *Proceedings of the 25th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '98)*, páginas 275–286. ACM, 1998.

[Dor10] Dormans, J. Adventures in level design: generating missions and spaces for action adventure games. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games (PCGames '10)*, artículo No. 1. ACM, 2010.

[DP10] Doran, J. y Parberry, I. Controlled procedural terrain generation using software agents. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):páginas 111–119, 2010.

[EBP+12] Emilien, A., Bernhardt, A., Peytavie, A., Cani, M.-P., y Galin, E. Procedural generation of villages on arbitrary terrains. *The Visual Computer*, 28(6-8):páginas 809–818, 2012.

[FFC82] Fournier, A., Fussell, D., y Carpenter, L. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):páginas 371–384, 1982.

[FFC09] Frade, M., Fernandéz de Vega, F., y Cotta, C. Breeding terrains with genetic terrain programming: the evolution of terrain generators. *International Journal of Computer Games Technology*, 2009, article ID 125714, 2009.

[FFC10a] Frade, M., Fernandéz de Vega, F., y Cotta, C. Evolution of artificial terrains for video games based on accessibility. *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation (EvoApplicatons '10)*, páginas 90–99. Springer, 2010.

[FFC10b] Frade, M., Fernandéz de Vega, F. d., y Cotta, C. Evolution of artificial terrains for video games based on obstacles edge length. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC '10)*, páginas 1–8. IEEE, 2010.

[Fin08] Finkenzeller, D. Detailed building facades. *IEEE Computer Graphics and Applications*, 28(3):páginas 58–66, 2008.

[Gal81] Galle, P. An algorithm for exhaustive generation of building floor plans. *Communications of the ACM*, 24(12):páginas 813–825, 1981.

[GGG+13] Génevaux, J. D., Galin, E., Guérin, E., Peytavie, A., y Beneš, B. Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics (TOG)*, 32(4), artículo No. 143, 2013.

[GMB06] Glass, K. R., Morkel, C., y Bangay, S. D. Duplicating road patterns in south african informal settlements using procedural techniques. *Proceedings of the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH '06)*, páginas 161–169. ACM, 2006.

[GMS09] Gain, J., Marais, P., y Straßer, W. Terrain sketching. *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (i3D '09)*, páginas 31–38. ACM, 2009.

[Gol89] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley Publishing Company, 1989.

[GPMG10] Galin, E., Peytavie, A., Maréchal, N., y Guérin, E. Procedural generation of roads. *Computer Graphics Forum*, 29(2):páginas 429–438, 2010.

[GPSL03] Greuter, S., Parker, J., Stewart, N., y Leach, G. Real-time procedural generation of 'pseudo infinite'cities. *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia (GRAPHITE '03)*, páginas 87–ff. ACM, 2003.

[GS09] Germer, T. y Schwarz, M. Procedural arrangement of furniture for real-time walkthroughs. *Computer Graphics Forum*, 28(8):páginas 2068–2078, 2009.

[Ham01] Hammes, J. Modeling of ecosystems as a data source for real-time terrain rendering. *Proceedings of the First International Symposium on Digital Earth Moving (DEM '01)*, páginas 98–111. Springer, 2001.

[Hay94] Haykin, S. *Neural Networks: A Comprehensive Foundation*. Prentice–Hall, 1994.

[HBW06] Hahn, E., Bose, P., y Whitehead, A. Persistent realtime building interior generation. *Proceedings of the 2006 ACM SIGGRAPH Symposium on Videogames*, páginas 179–186. ACM, 2006.

[HGA+10] Hnaidi, H., Guérin, E., Akkouche, S., Peytavie, A., y Galin, E. Feature based terrain generation using diffusion equation. *Computer Graphics Forum*, 29(7):páginas 2179–2186, 2010.

[HGS09] Hastings, E. J., Guha, R. K., y Stanley, K. O. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):páginas 245–263, 2009.

[HMVDVI11] Hendrikx, M., Meijer, S., Van Der Velden, J., y Iosup, A. Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications and Applications*, 2011.

[HNR68] Hart, P. E., Nilsson, N. J., y Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):páginas 100–107, 1968.

[Ios11] Iosup, A. Poggi: generating puzzle instances for online games on grid infrastructures. *Concurrency and Computation: Practice & Experience*, 23(2):páginas 158–171, 2011.

[JTSWF10] Jennings-Teats, M., Smith, G., y Wardrip-Fruin, N. Polymorph: Dynamic difficulty adjustment through level generation. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games (PCGames '10)*, artículo No. 11. ACM, 2010.

[JYT10] Johnson, L., Yannakakis, G. N., y Togelius, J. Cellular automata for real-time generation of infinite cave levels. *Proceedings of the 2010 Workshop*

on Procedural Content Generation in Games (PCGames '10), artículo No. 10. ACM, 2010.

[KBKŠ09] Krištof, P., Beneš, B., Křivánek, J., y Štáva, O. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum*, 28(2):páginas 219–228, 2009.

[KE81] Koning, H. y Eizenberg, J. The language of the prairie: Frank lloyd wright's prairie houses. *Environment and Planning B: Planning and Design*, 8(3):páginas 295–323, 1981.

[Kjø00] Kjølaas, K. A. H. *Automatic Furniture Population of Large Architectural Models*. Tesis Doctoral, Massachusetts Institute of Technology, Cambridge, Estados Unidos, 2000.

[KM07] Kelly, G. y McCabe, H. Citygen: An interactive system for procedural city generation. *Proceedings of the 5th International Conference on Game Design and Technology (GDTW '07)*, páginas 8–16. 2007.

[KMN88] Kelley, A. D., Malin, M. C., y Nielson, G. M. Terrain simulation using a model of stream erosion. *Proceedings of the 15th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '88)*, páginas 263–268. ACM, 1988.

[KU07] Kamal, K. y Uddin, Y. Parametrically controlled terrain generation. *Proceedings of the 5th international Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia (GRAPHITE '07)*, páginas 17–23. ACM, 2007.

[LD99] Lintermann, B. y Deussen, O. Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 19(1):páginas 56–65, 1999.

[LG06] Larive, M. y Gaildrat, V. Wall grammar for building generation. *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE '06)*, páginas 429–437. ACM, 2006.

[Lig00] Liggett, R. S. Automated facilities layout: past, present and future. *Automation in Construction*, 9(2):páginas 197–215, 2000.

[Lin68] Lindenmayer, A. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):páginas 280–299, 1968.

[LLRG04] Larive, M., Le Roux, O., y Gaildrat, V. Using meta-heuristics for constraint-based 3d objects layout. *Proceedings of the 2004 International Conference on Computer Graphics and Artificial Intelligence (3IA '04)*. 2004.

[LRW+06] Lechner, T., Ren, P., Watson, B., Brozefski, C., y Wilenski, U. Procedural modeling of urban land use. *ACM SIGGRAPH 2006 Research posters (SIGGRAPH '06)*, artículo No. 135. ACM, 2006.

[LTS+10] Lopes, R., Tutenel, T., Smelik, R. M., Kraker, K. J. de , y Bidarra, R. A constrained growth method for procedural floor plan generation. *Proceedings of the 11th International Conference on Intelligent Games and Simulation (GAME-ON '10)*, páginas 13–20. 2010.

[LWW03] Lechner, T., Watson, B., y Wilensky, U. Procedural city modeling. *Proceedings of the 1st Midwestern Graphics Conference*. 2003.

[LWW08] Lipp, M., Wonka, P., y Wimmer, M. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics (TOG)*, 27(3), artículo No. 102, 2008.

[Mar06] Martin, J. Procedural house generation: A method for dynamically generating floor plans. *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games (i3D '06)*. ACM, 2006.

[MCG03] Müller, M., Charypar, D., y Gross, M. Particle-based fluid simulation for interactive applications. *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation (SCA '03)*, páginas 154–159. Eurographics Association, 2003.

[MCP02] Michalek, J., Choudhary, R., y Papalambros, P. Architectural layout design optimization. *Engineering Optimization*, 34(5):páginas 461–484, 2002.

[MDH07] Mei, X., Decaudin, P., y Hu, B. G. Fast hydraulic erosion simulation and visualization on gpu. *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications (PG '07)*, páginas 47–56. IEEE, 2007.

[Mil86] Miller, G. The definition and rendering of terrain maps. *Proceedings of the 13th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '86)*, páginas 39–48. ACM, 1986.

[MKM89] Musgrave, F., Kolb, C., y Mace, R. The synthesis and rendering of eroded fractal terrains. *Proceedings of the 16th International Conference on*

Computer Graphics and Interactive Techniques (SIGGRAPH '89), páginas 41–50. ACM, 1989.

[MM10] Marson, F. y Musse, S. R. Automatic real-time generation of floor plans based on squarified treemaps algorithm. *International Journal of Computer Games Technology*, 2010, article ID 624817, 2010.

[MS05a] Mateas, M. y Stern, A. Procedural authorship: A case-study of the interactive drama façade. *Proceedings of Digital Arts and Culture (DAC)*. 2005.

[MS09] McCrae, J. y Singh, K. Sketch-based path design. *Proceedings of Graphics Interface 2009 (GI '09)*, páginas 95–102. Canadian Information Processing Society, 2009.

[MSK10] Merrell, P., Schkufza, E., y Koltun, V. Computer-generated residential building layouts. *ACM Transactions on Graphics (TOG)*, 29, artículo No. 181, 2010.

[Mus93] Musgrave, F. *Methods for Realistic Landscape Imaging*. Tesis Doctoral, Yale University, New Haven, Estados Unidos, 1993.

[MWH+06] Müller, P., Wonka, P., Haegler, S., Ulmer, A., y Van Gool, L. Procedural modeling of buildings. *ACM Transactions on Graphics (TOG)*, 25(3):páginas 614–623, 2006.

[MZWVG07] Müller, P., Zeng, G., Wonka, P., y Van Gool, L. Image-based procedural modeling of facades. *ACM Transactions on Graphics (TOG)*, 26(3), artículo No. 85, 2007.

[Nar04] Nareyek, A. Ai in computer games. *ACM Queue*, 1(10):páginas 58–65, 2004.

[Neu96] Neumann, J. V. *Theory of Self-Reproducing Automata*. University of Illinois Press Champaign, 1996.

[Ols04] Olsen, J. *Realtime procedural terrain generation*. Informe técnico, University of Southern Denmark, Department of Mathematics And Computer Science (IMADA), 2004.

[OSKL05] Ong, T. J., Saunders, R., Keyser, J., y Leggett, J. J. Terrain generation using genetic algorithms. *Proceedings of the 7th International Conference on Genetic and Evolutionary Computation (GECCO '05)*, páginas 1463–1470. ACM, 2005.

[Per85] Perlin, K. An image synthesizer. *Proceedings of the 13th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '85)*, páginas 287–296. ACM, 1985.

[PGGM09] Peytavie, A., Galin, E., Grosjean, J., y Merillou, S. Arches: a framework for modeling complex terrains. *Computer Graphics Forum*, 28(2):páginas 457–467, 2009.

[PH93] Prusinkiewicz, P. y Hammel, M. A fractal model of mountains with rivers. *Proceesings of Graphics Interface (GI '93)*, páginas 174–180. Canadian Information Processing Society, 1993.

[PL91] Prusinkiewicz, P. y Lindenmayer, A. *The algorithmic beauty of plants*. Springer, 1991.

[PM01] Parish, Y. I. y Müller, P. Procedural modeling of cities. *Proceedings of the 28th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, páginas 301–308. ACM, 2001.

[PMKL01] Prusinkiewicz, P., Mündermann, L., Karwowski, R., y Lane, B. The use of positional information in the modeling of plants. *Proceedings of the 28th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, páginas 289–300. ACM, 2001.

[PSK+12] Pirk, S., Stava, O., Kratt, J., Said, M. A. M., Neubert, B., Měch, R., Benes, B., y Deussen, O. Plastic trees: Interactive self-adapting botanical tree models. *ACM Transactions on Graphics (TOG)*, 31(4), artículo No. 50, 2012.

[PTY09] Pedersen, C., Togelius, J., y Yannakakis, G. N. Modeling player experience in super mario bros. *Proceedings of the 5th International Conference on Computational Intelligence and Games (CIG '09)*, páginas 132–139. IEEE, 2009.

[RCMLS96] Rau-Chaplin, A., MacKay-Lyons, B., y Spierenburg, P. The lahave house project: Towards an automated architectural design service. *Proceedings of the International Conference on Computer-Aided Design (CADEX '96)*, páginas 25–31. IEEE, 1996.

[Rey87] Reynolds, C. W. Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*, páginas 25–34. ACM, 1987.

[RFF10] Rodrigues, N., Frade, M., y Fernandéz de Vega, F. Development of chapas an open source video game with genetic terrain programming. *Actas del VII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB '10)*, páginas 135–142. 2010.

[RNC+96] Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M., y Edwards, D. D. *Artificial intelligence: a modern approach*. Prentice–Hall, 1996.

[RZL11] Raffe, W. L., Zambetta, F., y Li, X. Evolving patch-based terrains for use in video games. *Proceedings of the 13th International Conference on Genetic and Evolutionary Computation (GECCO '11)*, páginas 363–370. ACM, 2011.

[RZL12] Raffe, W. L., Zambetta, F., y Li, X. A survey of procedural terrain generation techniques using evolutionary algorithms. *Proceedings of 2012 IEEE Congress of Evolutionary Computation (CEC '12)*, páginas 10–15. IEEE, 2012.

[Sau06] Saunders, R. L. *Realistic terrain synthesis using genetic algorithms*. Tesis Doctoral, Texas A&M University, College Station, Estados Unidos, 2006.

[ŠBM+10] Štáva, O., Beneš, B., Měch, R., Aliaga, D. G., y Krištof, P. Inverse procedural modeling by automatic generation of l-systems. *Computer Graphics Forum*, 29(2):páginas 665–674, 2010.

[SBS94] Schwarz, A., Berry, D., y Shaviv, E. Representing and solving the automated building design problem. *Computer-Aided Design*, 26(9):páginas 689–698, 1994.

[SdKT+09] Smelik, R. M., Kraker, K. J. de , Tutenel, T., Bidarra, R., y Groenewegen, S. A. A survey of procedural methods for terrain modelling. *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS '09)*, páginas 25–34. 2009.

[SG71] Stiny, G. y Gips, J. Shape grammars and the generative specification of painting and sculpture. *Proceedings of 1971 International Federation for Information Processing Congress*, páginas 1460–1465. IFIP, 1971.

[SLRLG03] Sanchez, S., Le Roux, O., Luga, H., y Gaildrat, V. Constraint-based 3d-object layout using a genetic algorithm. *Proceedings of the 2003 International Conference on Computer Graphics and Artificial Intelligence (3IA '03)*. 2003.

[SP10] Sorenson, N. y Pasquier, P. Towards a generic framework for automated video game level creation. *Proceedings of the 2010 International Conference on Applications of Evolutionary Computation (EvoApplications '10)*, páginas 131–140. Springer, 2010.

[STdKB08] Smelik, R. M., Tutenel, T., Kraker, K. J. de , y Bidarra, R. A proposal for a procedural terrain modelling framework. *Poster Proceedings of the 14th Eurographics Symposium on Virtual Environments (EGVE '08)*, páginas 39–42. Eurographics Association, 2008.

[STWM09] Smith, G., Treanor, M., Whitehead, J., y Mateas, M. Rhythm-based level generation for 2d platformers. *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG '09)*, páginas 175–182. ACM, 2009.

[SYBG02] Sun, J., Yu, X., Baciu, G., y Green, M. Template-based generation of road networks for virtual city modeling. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '02)*, páginas 33–40. ACM, 2002.

[SYT10] Shaker, N., Yannakakis, G. N., y Togelius, J. Towards automatic personalized content generation for platform games. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE '10)*, páginas 63–68. AAAI, 2010.

[TBSdK09] Tutenel, T., Bidarra, R., Smelik, R. M., y Kraker, K. J. de . Rule-based layout solving and its application to procedural interior generation. *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS '09)*, páginas 15–24. 2009.

[TDNL07] Togelius, J., De Nardi, R., y Lucas, S. M. Towards automatic personalised content creation for racing games. *Proceedings of the 3th International Conference on Computational Intelligence and Games (CIG '07)*, páginas 252–259. IEEE, 2007.

[Teo08] Teoh, S. T. River and coastal action in automatic terrain generation. *Proceedings of the 2008 International Conference on Computer Graphics & Virtual Reality (CGVR '08)*, páginas 3–9. 2008.

[TPB+10] Togelius, J., Preuss, M., Beume, N., Wessing, S., Hagelbäck, J., y Yannakakis, G. N. Multiobjective exploration of the starcraft map space. *Pro-*

ceedings of the 6th International Conference on Computational Intelligence and Games (CIG '10), páginas 265–272. IEEE, 2010.

[TPY10] Togelius, J., Preuss, M., y Yannakakis, G. N. Towards multiobjective procedural map generation. *Proceedings of the 2010 Workshop on Procedural Content Generation in Games (PCGames '10)*, artículo No. 1. ACM, 2010.

[TW09] Tortelli, D. M. y Walter, M. Modeling and rendering the growth of speleothems in realtime. *Proceedings of International Conference on Computer Graphics Theory and Applications (GRAPP '15)*, páginas 27–35. 2009.

[TYSB11] Togelius, J., Yannakakis, G. N., Stanley, K. O., y Browne, C. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):páginas 172–186, 2011.

[VKW+12] Vanegas, C. A., Kelly, T., Weber, B., Halatsch, J., Aliaga, D. G., y Müller, P. Procedural generation of parcels in urban modeling. *Computer Graphics Forum*, 31(2):páginas 681–690, 2012.

[Vor08] Voronoï, G. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:páginas 198–287, 1908.

[Vos85] Voss, R. F. Random fractal forgeries. *Fundamental Algorithms for Computer Graphics*, páginas 805–835. Springer, 1985.

[WG10] Walsh, P. y Gade, P. Terrain generation using an interactive genetic algorithm. *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC '10)*, páginas 1–7. IEEE, 2010.

[WG11] Walsh, P. y Gade, P. The use of an aesthetic measure for the evolution of fractal landscapes. *Proceedings of the 2011 IEEE Congress on Evolutionary Computation (CEC '11)*, páginas 1613–1619. IEEE, 2011.

[WWSR03] Wonka, P., Wimmer, M., Sillion, F., y Ribarsky, W. Instant architecture. *ACM Transactions on Graphics (TOG)*, 22(3):páginas 669–677, 2003.

[YYT+11] Yu, L. F., Yeung, S. K., Tang, C. K., Terzopoulos, D., Chan, T. F., y Osher, S. J. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)*, 30(4), artículo No. 86, 2011.

[ZSTR07] Zhou, H., Sun, J., Turk, G., y Rehg, J. M. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):páginas 834–848, 2007.