Universidad de Concepción

Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Informática y Ciencias de la Computación

Implementación y comparación de distintos métodos de generación procedural para el juego FloorClearer

Memoria de Título

Autor: **Jesús Gómez Zambrano**
Profesor Guía: **Julio Godoy Del Campo**
Título al que opta: **Ingeniero Civil Informático**

Octubre 2024

Índice

1. Introducción

La industria del videojuego ha demostrado ser un negocio altamente lucrativo a nivel mundial. En 2023, las ganancias generadas se estiman en 343 mil millones de dólares globalmente, con proyecciones que sugieren un aumento de hasta 211 mil millones de dólares para 2029 [1]. Este crecimiento constante posiciona a los videojuegos como una de las formas de entretenimiento más rentables, superando incluso al cine y la música.

Una etapa fundamental en el desarrollo de videojuegos es la generación de contenido, que incluye la creación de personajes, escenarios y mundos. La generación procedural, utilizando algoritmos y reglas predefinidas, permite crear contenido dinámico y aleatorio, introduciendo variabilidad y rejugabilidad, además de reducir los tiempos de producción al automatizar parte del proceso, convirtiéndose en una herramienta invaluable.

En base a esto, en la presente memoria se investigarán e implementarán distintos métodos de generación de ambientes de manera procedural, aplicándolos en el contexto del juego FloorClearer. Este juego ha sido seleccionado por su estado de desarrollo temprano, así como por la facilidad de acceso a retroalimentación por parte del desarrollador. Se explorarán diversas posibilidades y se evaluarán los aspectos visuales, la coherencia y el rendimiento de cada método, con el objetivo de determinar las ventajas y desventajas de su aplicación en el juego mencionado.

1.1. Solución propuesta y alcances

Se plantea el investigar e implementar distintos métodos de generación de contenido procedural para la creación de entornos con el fin de evaluar cuál o cuáles de estos se adaptan mejor al juego objetivo, el cual entra dentro de la categoría de jugador contra entorno, donde este se enfrentará contra oleadas de enemigos intentando sobrevivir.

Con esto en mente, se estableció con el desarrollador del juego que los ambientes podían ser desde mazmorras hasta arenas o cuevas, definiéndose de la siguiente manera:

- **Mazmorras:** Entornos cerrados compuestos por una serie de habitaciones o cámaras conectadas por pasillos.
- **Arenas:** Áreas abiertas o semi-abiertas donde se llevan a cabo los combates entre el jugador y las oleadas de enemigos.
- **Cuevas:** Entornos subterráneos con una estructura menos lineal que las mazmorras, con pasajes irregulares y espacios abiertos mezclados con túneles estrechos.

En base a estas características, se deben buscar e implementar los métodos, considerando estas limitantes.

2. Objetivos

2.1. Objetivo General

El objetivo de esta memoria de título es implementar y comparar distintas técnicas de generación procedural de ambientes, evaluando sus aspectos funcionales y estéticos en el juego FloorClearer.

2.2. Objetivos Específicos

- Definir los requerimientos funcionales y no funcionales de los ambientes a generar con el desarrollador del juego FloorClearer.
- Investigar los métodos de generación procedural más adecuados para la tarea de generación de ambientes.
- Determinar métricas para comparar el desempeño entre los métodos de generación procedural seleccionados.
- Implementar los métodos seleccionados
- Comparar los resultados obtenidos por los distintos métodos utilizando las métricas establecidas.

3. Marco Teórico

La generación de contenido procedural es una herramienta útil durante el desarrollo de videojuegos, estos métodos de generación de contenido permiten dedicar una cantidad de recursos de tiempo y mano de obra significativamente menor que lo que se necesitaría en caso de hacerlo utilizando métodos tradicionales [2], lo que permite enfocar estos esfuerzos a otras áreas del proceso. El objetivo de esta sección es presentar las definiciones de términos y conceptos necesarios para la plena comprensión del contexto de la presente memoria, así como brindar ejemplos de trabajos relacionados con este.

3.1. Conceptos Previos

3.1.1. Juego de disparos en primera persona

Este género de juegos también llamados FPS por sus siglas en inglés es un tipo de juegos en los que el jugador ve el mundo a través de la perspectiva de uno de los personajes y debe atacar enemigos para avanzar.

3.1.2. FloorClearer

FloorClearer es un juego del género de disparos en primera persona, que toma inspiración de otros juegos y medios como lo son el Magika, Ultrakill y otros juegos del mismo género. Este se encuentra actualmente en estado de desarrollo por un estudiante de postgrado de la Universidad de Concepción. Este es el juego en el que se aplicarán las técnicas de generación de contenido procedural para crear los distintos escenarios en los que se desarrollará la partida.

La elección de aplicar las técnicas de generación procedural en FloorClearer deriva de varios factores como el temprano estado de su desarrollo, el fácil acceso al juego, la retroalimentación del creador al ser éste un estudiante de la Universidad de Concepción y finalmente a las características deseadas de los entornos a generar que permiten implementar una variedad de métodos sin estar restringidos a un aspecto específico.

3.1.3. Unity

Unity es un motor de juegos multiplataforma que se ha posicionado como uno de los más usados de la industria de los videojuegos. Permite la creación de juegos 2D y 3D, brindando a los desarrolladores herramientas como motor de físicas, sistema de scripting en C#, y acceso a recursos o componentes reutilizables que pueden incluir modelos 3D, texturas, sonidos, scripts, y otros elementos necesarios para el desarrollo de un videojuego. Este es el motor en el que se está desarrollando FloorClearer, por lo que las implementaciones realizadas en esta memoria se harán en este medio.

3.1.4. Generación de Contenido Procedural

La generación de contenido procedural (PCG, por sus siglas en inglés) es el uso de algoritmos para la creación de contenidos de un juego con una intervención indirecta o limitada por parte del usuario [3]. Este contenido puede incluir niveles, mapas, reglas, objetos, personajes, y más. La generación procedural es una técnica poderosa que permite crear experiencias únicas y variadas para los jugadores, reduciendo al mismo tiempo el esfuerzo y costo de desarrollo.

Existen varios tipos de algoritmos utilizados en la generación procedural, cada uno con sus propias ventajas y aplicaciones:

- **Algoritmos basados en búsqueda**: Estos algoritmos utilizan técnicas de búsqueda como búsqueda en profundidad, búsqueda en anchura, o algoritmos más avanzados para generar contenido que cumpla con ciertos criterios o metas [4]. Estos métodos son especialmente útiles para la generación de niveles que deben cumplir con ciertas restricciones o características específicas.

- **Generación basada en autómatas celulares**: Los autómatas celulares son utilizados para crear patrones y estructuras complejas a partir de reglas simples. Un ejemplo común es la generación de terrenos o biomas en juegos.

- **Generación basada en ruido o fractales**: Algoritmos como el ruido Perlin o el ruido Simplex [5] generan variaciones naturales y suaves, que son ideales para la creación de terrenos, texturas, y otros elementos que requieren un aspecto orgánico. Estos métodos pueden ser combinados con otras técnicas para añadir detalles y complejidad al contenido generado.

- **Algoritmos basados en machine learning**: Estos algoritmos aprovechan técnicas de aprendizaje automático para generar contenido que puede aprender y mejorar con el tiempo. Pueden usar redes neuronales y aprendizaje profundo para analizar grandes cantidades de datos de juego y generar contenido que se adapte a los patrones y preferencias del jugador o del desarrollador. Un ejemplo es el uso de redes generativas adversarias (GANs) [6] para crear niveles de juego nuevos y originales basados en ejemplos previos.

La generación procedural ofrece numerosas ventajas, tales como:

- **Rejugabilidad aumentada**: Al generar contenido dinámicamente, cada partida puede ofrecer una experiencia diferente, aumentando la rejugabilidad.

- **Escalabilidad**: Permite crear grandes cantidades de contenido con menos esfuerzo, lo cual es crucial para juegos con mundos amplios.

- **Adaptabilidad**: El contenido puede ser ajustado en base a las necesidades del juego, modificando parámetros o valores de los métodos.

3.2. Trabajos relacionados

En el área de PCG, ha habido desarrollo significativo con múltiples enfoques y metodologías. A continuación, se presentan algunos de los trabajos relevantes en esta área.

En [7], se aborda la generación de contenido procedural y se describen los distintos métodos que se usan para este problema. Este trabajo se utilizó como base para conocer estas técnicas, así como sus características y usos que se les dan generalmente.

En[8], se describe la utilización de "Content Chunks", una estructura propuesta para enfocarse en los problemas de "abstracción" y "replicación" al momento de generar habitaciones. La abstracción es la selección de un objeto a generar dado un grupo de posibles objetos, y la replicación es la repetición de varios objetos de la misma selección. Esta idea puede servir para definir reglas que permitan una mejor coherencia al seleccionar objetos o estructuras para ubicarlas dentro de las arenas a generar.

En [9], se presenta un análisis de las mazmorras generadas proceduralmente de varios juegos, detallando el tipo de algoritmo usado, el género del juego y características como las dimensiones del juego o la generalidad de la solución. El estudio planteó observaciones como la tendencia al uso de soluciones basadas en búsqueda y sirvió como orientación en cuanto al uso de distintos métodos de PCG para distintos tipos de entornos.

En [3], se exploran en mayor profundidad los métodos tratados en [7], proporcionando más ejemplos y dedicando secciones completas a la representación del contenido para los métodos basados en búsqueda y a la construcción de niveles y mazmorras. De este libro se obtuvieron los tipos de métodos implementados en la presente memoria.

En base a esta revisión bibliográfica, podemos clasificar los métodos de PCG en:

- Métodos basados en búsqueda
- Algoritmos basados en autómatas celulares
- Algoritmos basados en ruido y fractales

3.2.1. Métodos basados en búsqueda

Los métodos basados en búsqueda asumen que existe una solución adecuada dentro del espacio de soluciones posibles, donde una solución se define como una configuración del entorno o contenido que cumple con los criterios y restricciones establecidas por el algoritmo. A través de ajustes e iteraciones, estos métodos buscan llegar a una solución óptima que satisfaga los objetivos y requisitos definidos. Los componentes de un método basado en búsqueda son:

- El algoritmo de búsqueda
- Alguna forma de representar el contenido a generar

- Función o funciones de evaluación.

Los algoritmos de búsqueda usados en el contexto de generación procedural de contenido suelen ser metaheurísticas como algoritmos evolutivos, donde se comienza con genotipos que representan un conjunto de soluciones iniciales y se iteran sobre ellos hasta obtener una solución aceptable o hasta una condición de tiempo definida.

La representación del contenido puede variar desde muy directa, como representar cada casilla del mapa como una posición de un arreglo y el número en esa posición representaría lo que habrá en esa casilla, hasta muy indirecta, como una semilla aleatoria que determinará las condiciones iniciales del algoritmo, o algo intermedio como objetos prefabricados reusables y una lista de cómo se distribuyen.

La función de evaluación determina la correctitud del algoritmo de búsqueda, reflejando que tan bien el algoritmo cumple con distintos criterios como jugabilidad, resolubilidad o estética, esta función puede ser:

- Directa: No se simula el gameplay durante la evaluación, es rápida y fácil de implementar, pero usualmente difícil de idear.
- Simulation-Based: Se usan agentes de IA para navegar el terreno generado y estimar su calidad.
- Interactiva: Se evalúa el resultado en base a las interacciones de humanos con este. Puede ser implícita o explícita.

Este método puede ser usado para generar cualquier tipo de contenido para juegos, pero dependiendo del contenido deseado puede llegar a ser lento y el tiempo que tarda en generar una buena solución en caso de funciones de evaluación más complejas no puede ser previsto fácilmente.

3.2.2. Algoritmos basados en autómatas celulares

Los autómatas celulares son representaciones de células colocadas en una grilla que cambian paso a paso en base a una serie de reglas, las que dependen del estado actual de cada célula y sus vecinos. Estas reglas se aplican de manera iterativa por una cantidad definida de repeticiones. Estos algoritmos han sido ampliamente usados en el desarrollo de videojuegos y, debido a su simplicidad, son altamente veloces, permitiendo crear contenido en tiempo de ejecución. Sin embargo, por esta misma simplicidad no aseguran jugabilidad ni resolubilidad. Estos pueden ser usados para generar, por ejemplo, la estructura general de una mazmorra (Figura 1), para luego usar esta misma como base inicial para otro tipo de algoritmo de PCG que se encargue de asegurar la resolubilidad de esta, o en una escala más pequeña, puede ser usado para generar patrones de objetos recolectables coherentes que sigan una serie de reglas simples, estos patrones después podrán ser usados por otros algoritmos para seleccionarlos y ubicarlos por el mapa.

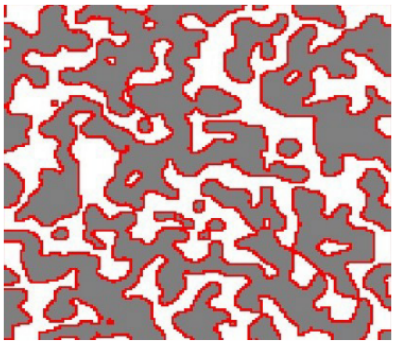

Figura 1: Ejemplo de cueva creada con autómatas celulares [3]

3.2.3. Algoritmos basados en ruido y fractales

Los algoritmos basados en ruido y fractales son técnicas matemáticas utilizadas para generar patrones complejos y naturales, como relieves, terrenos o texturas, de manera procedural. Estos algoritmos utilizan en funciones de ruido, como el ruido Perlin o Simplex, y procesos fractales, creando estructuras auto-similares y detalladas en diferentes escalas.

Este tipo de algoritmos son ampliamente usados para la generación de terreno, específicamente para el caso de terreno en 3D existen métodos como el "noise mesh" en el cual se crea terreno a partir de un mapa de altura generado en base a una función de ruido 2D. Por otro lado, en casos donde no es posible el uso de "heightmaps" se pueden emplear "voxels", los cuales son una representación en base a cubos del terreno.

Estos algoritmos pueden ser usados como base para generar terrenos irregulares en el caso de la generación de arenas, donde se emplearían para generar el suelo. Como alternativa, en [10] se propone un método de generación de mazmorras basado en fractales, por lo que podría ser usado directamente para generar entornos que coincidan con los requerimientos del cliente para el juego (Figura 2).

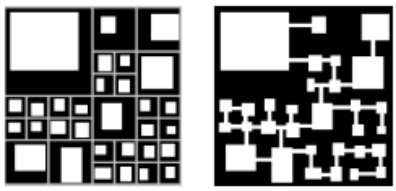

Figura 2: Ejemplo de mazmorra creada usando el método fractal de segmentación [3]

4. Métodos desarrollados

Tras realizar una investigación bibliográfica se seleccionaron e implementaron distintos métodos de generación procedural los cuales se detallarán a continuación.

4.1. Método basado en autómatas celulares

El método basado en autómatas celulares [11][12] permite generar entornos del tipo cueva sencillos utilizando reglas simples. Este algoritmo se separa en varias etapas, inicialmente genera una grilla en donde cada celda de esta puede ser o no una pared. La generación se realiza en base a una semilla y representa el entorno inicial (Figura 3), luego se definen las reglas para el autómata donde, usando una vecindad de Moore, la cual corresponde al conjunto de las 8 celdas que rodean a la original, cada celda que tenga al menos 4 paredes en su vencidad, se convierte en pared, si tiene 4 paredes exactamente, se mantiene en su estado actual y si tiene menos de 4 se convierte en espacio vacio (Figura 4). Esta regla simple se aplica en varias iteraciones donde tanto la cantidad de iteraciones como la cantidad de celdas requeridas para aplicar la regla mencionada anteriormente son parametros modificables a nivel de la interfaz de Unity.

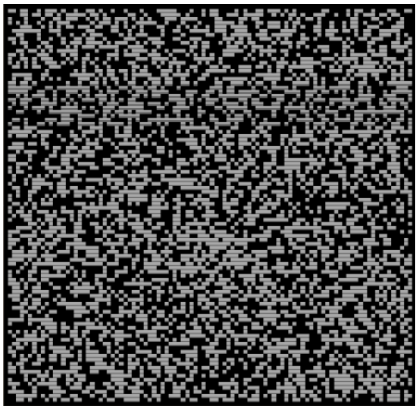

Figura 3: Generación de la grilla inicial

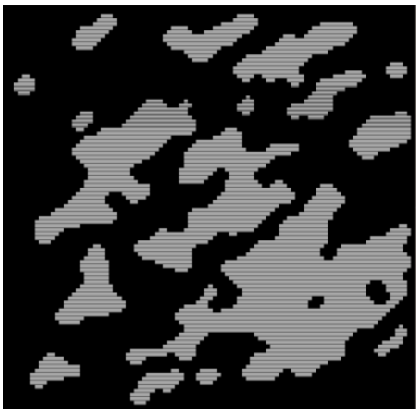

Figura 4: Suavizado tras aplicar iteraciones de la regla

Posteriormente, teniendo ya una grilla con aspecto de cueva gracias al paso anterior, se aplica un suavizado al tener definidos desde antes lospuntos medios en cada arista de los cuadrados de la mesh y todos los posibles triángulos que se pueden generar dependiendo de que puntos estén como pared y cuáles no, luego aplicando este diccionario de triángulos a toda la grilla se logra remover el aspecto cuadriculado que tenía anteriormente, otorgándole uno más cercano a otras cuevas de videojuegos (Figura 5).

Luego de esto, se hace una limpieza de todas las habitaciones o paredes que sean muy pequeñas, para producir un espacio más cohesivo (Figura 6).

Para lograr esto se recorren y almacenan en 2 arreglos todas las paredes y habitaciones, donde antes de almacenarse, se revisa que estén por encima de un umbral.

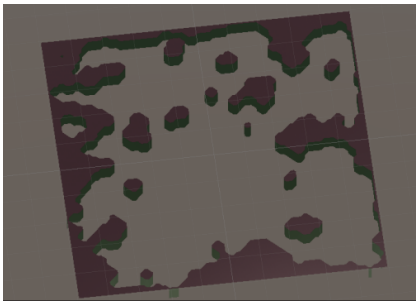

Figura 5: Suavizado usando los triángulos de Unity

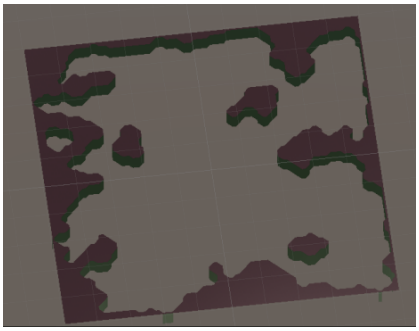

Figura 6: Limpieza de habitaciones y muros innecesarios

Tras esta eliminación, se inicia el proceso de la creación de pasillos entre habitaciones. Esta comienza enlazando cada habitación a su vecina más cercana, luego, definimos la habitación más grande como "habitación central" y separamos a todas las demás en 2 listas, las que pueden llegar a la habitación central y las que no, y se van enlazando los grupos de habitaciones más cercanos entre sí. Posteriormente, se actualizan las listas hasta que el arreglo de habitaciones que no pueden llegar a la central esté vacío. Finalmente se utiliza el algoritmo de línea de Bresenham, el cual es un método que determina el camino a tomar para dibujar los pasillos que representaban estas conexiones entre las habitaciones, y el código termina su ejecución(Figuras 7 y 8).

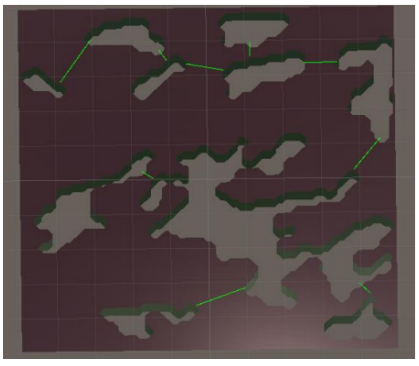

Figura 7: Conexión entre salas

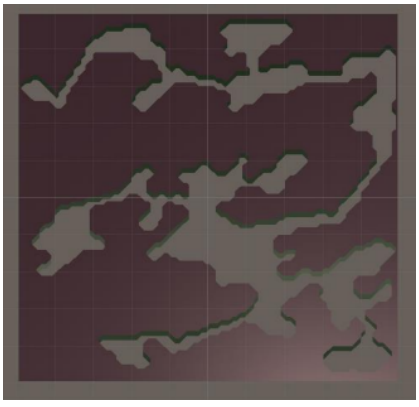

Figura 8: Resultado final tras generar pasillos usando Bresenham

En cuanto a las ventajas de este método se encuentran: la velocidad de

generación, pues al estar basado en una regla simple, puede generar cuevas en tiempo en el orden de milisegundos; la capacidad de control, al poder modificar los parámetros de cuantos vecinos se necesitan para cambiar a muro o espacio vacío; y también destaca que este método asegura resolubilidad gracias a los últimos pasos que generan las conexiones y los pasillos.

En contraparte, el método presentado muestra desventajas como la poca variabilidad de las cuevas debido a la naturaleza de la regla de generación, a su vez, la generación de los pasillos que conectan las habitaciones se ve cuadriculada y poco natural, pero esto podría solucionarse aplicando algún otro algoritmo de suavizado después de generar estos pasillos.

4.2. Método basado en Machine Learning: *Wave Function Collapse*

Este es un método iterativo basado en el concepto del colapso de función de onda (WFC por sus siglas en inglés), el cual ocurre cuando se fuerza a una función de onda a reducirse a un solo estado debido a la interacción con el mundo externo. En este caso, el método consta de tener un conjunto de partes (Figuras 9 y 10) de lo que se desea generar, que en este contexto corresponde a piezas de mazmorra, las cuales tendrán posibles vecinos en cada dirección, y en base a estas piezas y listas de posibles vecinos se genera el contenido deseado.

El algoritmo [15] sigue varias etapas. Primero, se genera una grilla sobre la cual se construirá el entorno. En cada casilla de esta, inicialmente se pueden generar cualquiera de las posibles piezas. En la primera iteración, se elige una casilla al azar, y se colapsa a una de sus piezas permitidas. Dado que aún no se han aplicado restricciones, en esta primera instancia, cualquiera de las piezas existentes puede ser seleccionada. Tras haber "colapsado" esa primera casilla de la grilla, actualizamos todas las casillas, removiendo las piezas que no cumplan con las reglas de adyacencia de la pieza recién colapsada, luego de esto, colapsamos la casilla con la menor cantidad de piezas permitidas, eligiendo al azar en caso de empate, y elegimos una pieza al azar de entre las disponibles después de aplicar las restricciones. Finalmente repetimos el paso de propagar las restricciones tras colapsar la casilla y elegir la casilla con menos piezas permitidas para ser colapsada hasta que todas las casillas de la grilla tengan una pieza asignada.

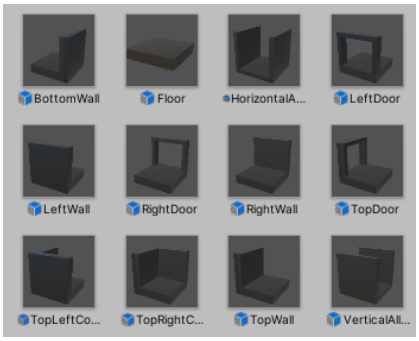

Figura 9: Algunas de las piezas usadas

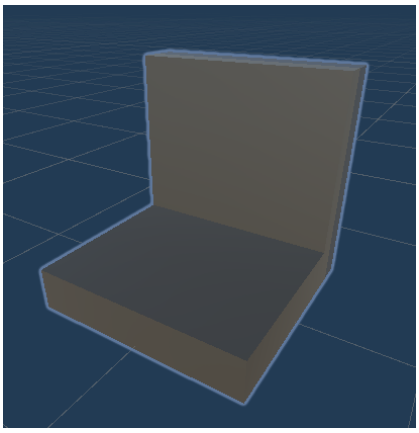

Figura 10: Acercamiento a la pieza "TopWall"

Entre las ventajas de este método está su posibilidad de personalización. Al utilizar piezas creadas a mano, permite generar tantos entornos como combinaciones de piezas se creen. El límite de la diversidad de los entornos no se encuentra en el algoritmo, sino en la cantidad y variedad de las piezas creadas para el mismo. Incluso es posible generar decoración dentro de las habitaciones con el mismo método, solo requiriendo que los elementos a decorar se modelen siguiendo las mismas reglas de las piezas mencionadas anteriormente.

Sin embargo, este método presenta varios aspectos negativos que lo convierten en una opción poco atractiva para el desarrollo de mazmorras. El primer inconveniente es el alto tiempo de generación. Se acordó con el stakeholder que los entornos debían generarse en tiempo de ejecución. No obstante, al intentar generar niveles del tamaño estándar de prueba, comparable con otros métodos, este requería varios minutos para completarse.

El problema radica en que, después de colapsar la primera casilla, es necesario recorrer toda la grilla actualizando las posibles piezas para cada casilla. Esto se realiza en función de las reglas de adyacencia de la pieza colocada y las restantes posibles, lo que implica un alto costo computacional. Como resultado, el tiempo de generación aumenta sustancialmente a medida que crece el tamaño del entorno. Este incremento es especialmente notable al ejecutarlo en el tamaño predeterminado usado para los métodos basados en grillas, donde la capacidad computacional disponible no es suficiente para completar el proceso.

En el mismo orden de ideas, otra de las deficiencias del WFC es el poco control que se tiene sobre el contenido generado. Al basarse únicamente en las listas de adyacencia, elementos como la cantidad de habitaciones, su tamaño y, más importante, las conexiones entre ellas no se pueden configurar ni asegurar.

Como resultado, este algoritmo comúnmente genera habitaciones aisladas sin conexión entre ellas (Figura 11). En casos más raros, se pueden observar habitaciones con múltiples pasillos que salen de ellas en formas antinaturales

(Figura 12). Además, es frecuente encontrar habitaciones muy pequeñas o sectores incompletos debido a intentos fallidos de generarlos cerca de los bordes del entorno, lo que en general se traduce en niveles no recorribles e inconsistentes.

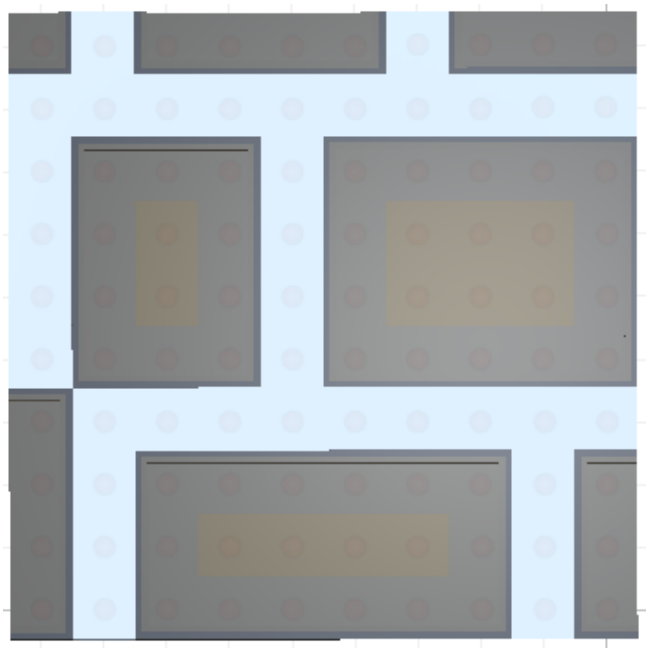

Figura 11: Resultado final con habitaciones desconectadas

Figura 12: Ejemplos de desconexiones y múltiples conexiones en implementaciones de la literatura

Finalmente, el último problema detectado con el WFC es que cada pieza debe ser modelada y etiquetada a mano, teniendo que llenar su lista de adyacencia con los posibles vecinos correspondientes, este proceso a pesar de ser sencillo, conlleva mucho tiempo y puede generar errores fatales en caso de que falte o sobre alguna pieza en alguna lista, además de que en caso de querer cambiar las piezas por otras para cambiar el aspecto o estructura del nivel, se deben

etiquetar las nuevas piezas desde 0, haciendo que a pesar de que el código es sencillo de implementar, se pierda mucho tiempo y recursos en este etiquetado de datos.

4.3. Método basado en búsqueda: *Depth First Search*

Este método [13][14] se basa en una de las formas más conocidas de recorrer grafos: la búsqueda en profundidad (DFS, por sus siglas en inglés). En DFS, se comienza en un nodo inicial y se exploran sus vecinos, profundizando en cada uno de ellos hasta el final antes de retroceder. Esta técnica se utiliza para generar contornos y habitaciones de mazmorras, aplicando la misma lógica para decidir qué áreas del entorno se expanden a continuación.

La generación se realiza utilizando objetos prefabricados, como habitaciones con pasillos ya construidos (Figura 13). Primero, se crean las habitaciones manualmente; en este caso, se utilizó una sala con cuatro salidas y, por defecto, dos pasillos que salen de las puertas. Estos pasillos pueden o no estar activos, dependiendo de si la habitación colinda con otra en esa dirección. Luego, se genera una grilla de un tamaño configurable por el usuario y se elige al azar una casilla de la grilla como nodo final, junto con una casilla inicial para comenzar a colocar las habitaciones prefabricadas.

Una vez completada la inicialización, se realiza una búsqueda en profundidad para intentar llegar al nodo declarado como nodo final. Durante este proceso, se avanza de nodo en nodo, colocando una habitación en cada uno de ellos y activando los pasillos correspondientes a la conexión entre la sala prefabricada recién colocada y la anterior.

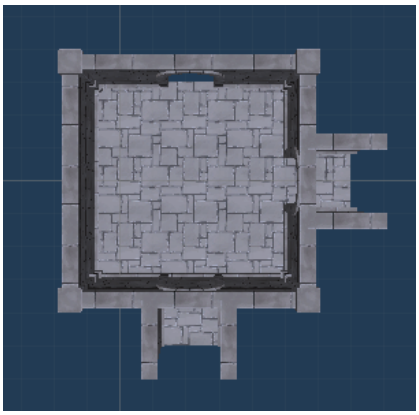

Figura 13: Habitación prefabricada para el DFS

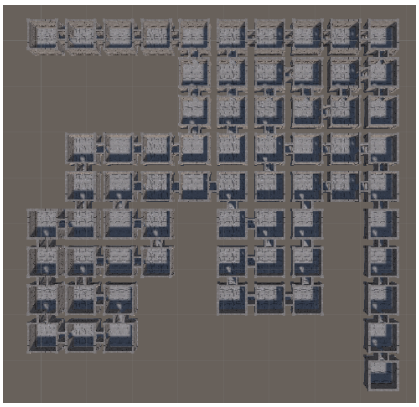

Figura 14: Resultado final del algoritmo

Entre las ventajas de este método se encuentran la facilidad de implementación, al estar basado en una técnica básica para recorrer grafos, el algoritmo que implementa este método es sencillo y fácil de entender. También permite

generar una variedad de habitaciones con la única limitación de que deben ser hechas a mano y que deben tener el mismo tamaño, sin embargo, aún con estas restricciones, se pueden crear mazmorras diversas.

Por otra parte, debido a la misma simplicidad del algoritmo, a pesar de que se pueda agregar variedad usando distintos modelos de habitación, el layout generado siempre sigue el patrón de un DFS convencional por lo que puede ser repetitivo o verse poco natural. Así mismo, como se mencionó anteriormente, las habitaciones deben ser todas del mismo tamaño, así que en caso de desear distintos tamaños de salas, este método no es el indicado pues para permitir eso, se necesitarían modificaciones substanciales. Finalmente, el método al ser tan sencillo no permite variar los parámetros más allá del ancho y largo de la mazmorra a generar, así como si se desea llenar la grilla completa o si existirá un nodo final, como se explicó con anterioridad.

4.4. Método basado en búsqueda: *Procedural Level Generator*

Este método usa un código de la asset store de Unity[16], el cuál similar al de DFS utiliza habitaciones prefabricadas. Sin embargo, este no tiene la limitación de que deban ser del mismo tamaño, pudiendo usar incluso objetos prefabricados más inusuales como segmentos de cuevas como se muestra en las imágenes promocionales del asset (Figura 15), proveyendo así una mayor variedad potencial de entornos en comparación a métodos similares que utilizan habitaciones prefabricadas.

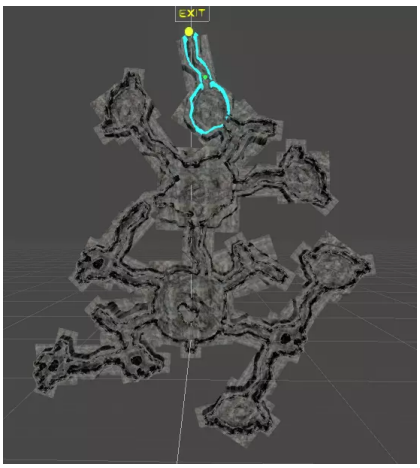

Figura 15: Cueva de ejemplo generada usando el Asset

Este asset funciona mediante un sistema de etiquetas, donde cada pieza prefabricada debe estar etiquetada para indicar su tipo y sus puntos de conexión.

Cada pieza necesita identificar todos los lugares donde puede conectarse con otras y el tipo de piezas con las que puede conectar. Por ejemplo, una habitación debe tener la etiqueta "habitación" y debe marcar todas sus salidas, además de poder conectarse únicamente con piezas del tipo pasillo.

Las piezas pueden tener múltiples etiquetas, permitiendo que cumplan diversas funciones. Por ejemplo, una habitación puede tener simultáneamente las etiquetas "habitación" y "habitación de jefe". Esto significa que un pasillo etiquetado como "pasillo" podrá conectarse con esta habitación, y también se podrán aplicar reglas especiales, como la restricción de que debe haber solo una habitación con la etiqueta "habitación de jefe".

En cuanto al algoritmo, este coloca una pieza inicial y comienza a posicionar piezas en los puntos de salida de esta, revisa si la pieza recién colocada colisiona con alguna de las ya colocadas, si es así, la retira y selecciona un nuevo punto de salida entre los disponibles para colocar una nueva pieza seleccionada al azar entre las compatibles con dicho punto. Este proceso se repite hasta completar el número de piezas solicitado en la interfaz. Antes de terminar, se revisa que todas las reglas especiales se cumplan, por ejemplo, si debe existir una sola pieza con la etiqueta "habitación del tesoro" y no ha sido creada, el algoritmo remueve la última pieza colocada e intenta integrar la faltante para cumplir las reglas, este paso se repite hasta que todas las reglas especiales se cumplan.

En cuanto a las ventajas que acarrea este algoritmo, prima la capacidad de personalización, en contraste con otros algoritmos similares que hacen uso de habitaciones y pasillos prefabricados (Figura 16). Además, este método no tiene limitaciones en cuanto al tamaño o la estética de las habitaciones generadas. Puede combinarse con el método de generación de habitaciones decoradas, utilizando estas como objetos prefabricados para el generador. También es posible diseñar las habitaciones manualmente, siguiendo las reglas de etiquetado de tipo y señalización de salidas. Esto permite que la variedad en la mazmorra dependa de la cantidad y diversidad de habitaciones que se proporcionen al algoritmo como piezas.

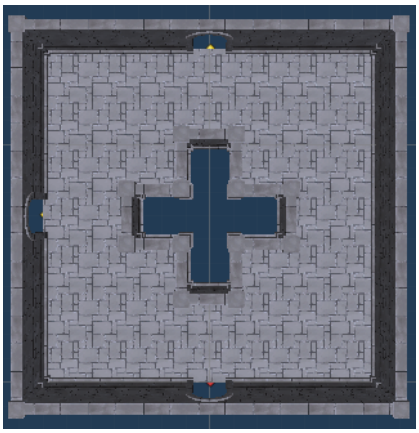

Figura 16: Ejemplo de habitación prefabricada

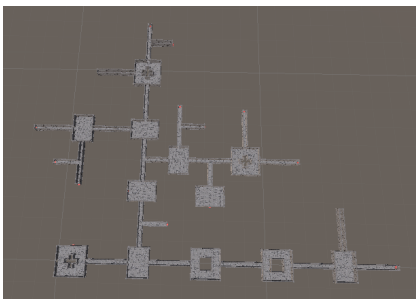

Figura 17: Resultado final del algoritmo

Otro punto positivo es la simplicidad del algoritmo y su fácil integración una vez que se tienen las piezas etiquetadas. Además de contar con un tutorial en video sobre cómo utilizar el asset, también se incluye un archivo README en PDF y un tutorial detallado sobre cómo implementar el algoritmo desde cero. Todo esto facilita enormemente la tarea de integración, quedando como el trabajo más laborioso la generación manual de las piezas.

Por otra parte, entre los aspectos negativos, el algoritmo al solo ir adjuntando piezas hasta que llega a la cantidad deseada suele generar muchos pasillos que no llevan a ningún lado empeorando el aspecto final de la mazmorra, y al ser un algoritmo tan simple y no llevar control de las conexiones, complica el poder limpiar estos pasillos para dejar solo los que conecten con habitaciones en todas sus salidas. Del mismo modo puede posicionar habitaciones con múltiples salidas, en las cuales no todas las salidas llevan a algún sitio, generando una mazmorra recorrible pero inconsistente.

4.5. Método basado en fractales: *Binary Space Partitioning*

El método basado en fractales implementado es el de "Partición binaria del espacio" (BSP por sus siglas en inglés[17][18]). Este método permite generar mazmorras con habitaciones rectangulares en base a parámetros simples como el tamaño de la mazmorra, el de los pasillos, el de las habitaciones, la separación entre estas últimas y el número de iteraciones que realizará el algoritmo.

El modo en que funciona es que inicialmente se tiene un espacio completo, se decide entre hacer un corte horizontal o vertical que divida este espacio en dos partes. Estas serán los espacios donde se generen las habitaciones por lo que se verifica que las salas resultantes cumplan con los parámetros de ancho y alto mínimo, en caso de incumplir alguna de estas restricciones, se repite el paso de

seleccionar una dirección y un punto para dividir el espacio en dos.

Tras la iteración inicial, al quedar más de una sección, se selecciona una entre las generadas y esa es la que se divide en dos partes nuevas, llevando en paralelo la referencia de cómo se dividen usando como estructura un árbol binario. Este proceso se repite hasta que no se pueda seccionar más el terreno sin incumplir las reglas del tamaño de la habitación o hasta que se cumplan las iteraciones establecidas en la interfaz.

Posteriormente, se procede a generar las habitaciones en los espacios que el algoritmo ha seccionado, asignando una habitación por sección. Una vez que todas las habitaciones han sido establecidas, se utiliza el árbol binario que referencia las separaciones para decidir qué habitaciones estarán conectadas entre sí mediante pasillos. En este caso, todas las habitaciones que sean hijas del mismo nodo del árbol estarán conectadas. En situaciones donde hay conjuntos de habitaciones, se conectan aquellas que están más cerca entre sí dentro de cada conjunto (Figura 18)

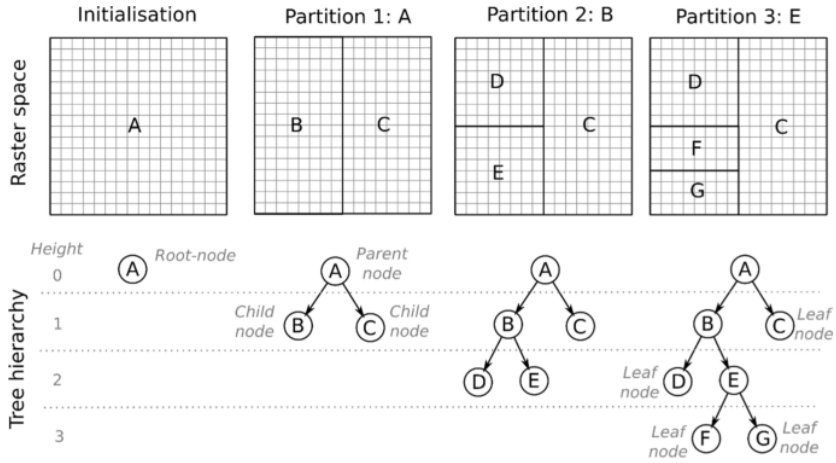

Figura 18: Diagrama del funcionamiento del BSP [5]

Este algoritmo tiene entre sus ventajas la velocidad de generación, al poder manejar el crear mazmorras de tamaño elevado de manera casi instantánea, así como el garantizar una mazmorra recurrible y con conexiones coherentes entre habitaciones.

Sin embargo, en contraposición a otros métodos, este método no permite tanta variabilidad en las mazmorras generadas, siempre generando habitaciones rectangulares sin mucha diferencia entre ellas, otro punto negativo es que al elegirse la dirección en que se secciona el terreno de manera aleatoria, es posible que se seccione varias veces en la misma dirección quedando múltiples habitaciones alargadas generando una mazmorra con un aspecto poco natural (Figura 19).

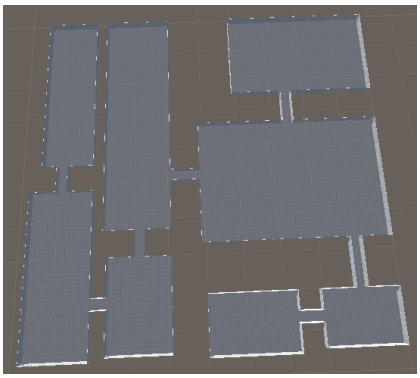

Figura 19: Layout resultante tras finalizar el algoritmo

5. Resultados

Las Tablas de 1 a 5 muestran la evaluación dada por el cliente de los diferentes métodos de generación procedural implementados, así como de diferentes encuestados. Esta evaluación está basada en un conjunto de parámetros decididos con el desarrollador del juego.

Descripción de los Parámetros

- **Adaptabilidad (1-10):** Evalúa qué tan compatible es de utilizar el método con el juego, considerando factores como la ausencia de errores y la eficiencia en la generación de contenido. Este parámetro solo se le solicitó al desarrollador, pues es el único que cuenta con contexto suficiente del juego en cuestión como para poder evaluar este campo.
- **Aspecto (1-5):** Valora la estética del contenido generado. Para métodos con prefabs hechos a mano, se considera el diseño general en lugar de los assets específicos.
- **Coherencia (1-5):** Mide la consistencia del contenido generado dentro del contexto del juego y entre sí.
- **Facilidad de Integración (FI) (1-5):** Indica qué tan sencillo es integrar el método en el juego existente. Este parámetro solo se le solicitó al desarrollador por los mismos motivos expuestos en el caso de adaptabilidad.
- **Facilidad de Uso (FU) (1-5):** Considera la comodidad de uso de la herramienta, incluyendo la calidad de la documentación y la facilidad de comprensión.

Tablas de Evaluación

Método	**Adaptabilidad**	**Aspecto**	**Coherencia**	**FI**	**FU**	**Total**
Desarrollador de FloorClearer	1	1	1	1	2	6
Encuestado 1	N/A	2	1	N/A	2	5
Encuestado 2	N/A	2	1	N/A	1	4
Encuestado 3	N/A	1	1	N/A	3	5
Encuestado 4	N/A	1	1	N/A	2	4
Encuestado 5	N/A	1	1	N/A	2	4
Promedio	**1**	**1.33**	**1**	**1**	**2**	**4.66**

Tabla 1: Resultados de la evaluación del método WFC

Método	**Adaptabilidad**	**Aspecto**	**Coherencia**	**FI**	**FU**	**Total**
Desarrollador de FloorClearer	8	5	4	5	2	24
Encuestado 1	N/A	3	5	N/A	2	10
Encuestado 2	N/A	5	5	N/A	1	11
Encuestado 3	N/A	4	4	N/A	3	11
Encuestado 4	N/A	4	5	N/A	4	13
Encuestado 5	N/A	5	5	N/A	3	13
Promedio	**8**	**4.33**	**4.67**	**5**	**2.5**	**13.6**

Tabla 2: Resultados de la evaluación del método DFS

Método	**Adaptabilidad**	**Aspecto**	**Coherencia**	**FI**	**FU**	**Total**
Desarrollador de FloorClearer	10	5	5	5	4	29
Encuestado 1	N/A	3	5	N/A	2	10
Encuestado 2	N/A	5	5	N/A	2	12
Encuestado 3	N/A	4	4	N/A	4	12
Encuestado 4	N/A	5	5	N/A	5	15
Encuestado 5	N/A	3	4	N/A	4	11
Promedio	**10**	**4.17**	**4.67**	**5**	**3.50**	**14.83**

Tabla 3: Resultados de la evaluación del método Procedural Level Generator

Método	**Adaptabilidad**	**Aspecto**	**Coherencia**	**FI**	**FU**	**Total**
Desarrollador de FloorClearer	8	4	3	2	2	19
Encuestado 1	N/A	4	4	N/A	2	10
Encuestado 2	N/A	3	3	N/A	1	7
Encuestado 3	N/A	4	3	N/A	3	10
Encuestado 4	N/A	5	5	N/A	3	13
Encuestado 5	N/A	3	4	N/A	3	10
Promedio	**8**	**3.83**	**3.67**	**2**	**2.33**	**11.5**

Tabla 4: Resultados de la evaluación del método BSP

Método	**Adaptabilidad**	**Aspecto**	**Coherencia**	**FI**	**FU**	**Total**
Desarrollador de FloorClearer	8	1	3	4	1	17
Encuestado 1	N/A	3	2	N/A	3	8
Encuestado 2	N/A	4	3	N/A	2	9
Encuestado 3	N/A	3	4	N/A	4	11
Encuestado 4	N/A	3	4	N/A	4	11
Encuestado 5	N/A	3	3	N/A	3	9
Promedio	**8**	**2.83**	**3.50**	**4**	**2.83**	**10.8**

Tabla 5: Resultados de la evaluación del método basado en Autómatas Celulares

La Tabla 6 muestra el desempeño de tiempo de los métodos implementados, todos fueron evaluados usando mazmorras del mismo tamaño, sin embargo, no se pudieron obtener resultados del algoritmo Wave Function Collapse para esas dimensiones debido a limitaciones computacionales. A pesar de esto, para entornos más reducidos, este método puede tardar varios minutos en generar el resultado completo, por lo que no es aconsejable usarlo en juegos que requieran generación a tiempo de ejecución.

Procedural Level Generator	**DFS**	**BSP**	**Autómata Celular**
64	16	30	214
74	13	23	213
66	17	25	223
44	8	23	216
64	10	25	234
65	13	28	222
62	17	29	231
67	22	25	218
59	13	25	214
71	20	27	217
63.6	**14.9**	**26**	**220.2**

Tabla 6: Resultados de tiempo para los diferentes métodos (ms)

Los perfiles de las personas encuestadas varían desde individuos sin conocimientos en informática y con poca experiencia en videojuegos, hasta estudiantes y graduados en informática con experiencia en el desarrollo de videojuegos. Se observó un patrón en el que el grupo sin experiencia evaluó con baja puntuación la facilidad de uso de todos los métodos, debido a su limitado conocimiento sobre el uso de Unity. En contraste, el grupo con experiencia y el desarrollador del juego no presentaron este sesgo. No se identificaron patrones similares en otros parámetros evaluados.

Método basado en autómatas celulares

El método de *Autómatas Celulares* presenta una adaptabilidad alta (8/10) debido a su capacidad para generar mapas sin errores significativos en la colocación de elementos. Sin embargo, la evaluación del desarrollador del aspecto visual es pobre (1/5), posiblemente debido a la falta de detalles gráficos o prefabs de alta calidad así como errores al exportar las texturas de la cueva como tal, limitando el aspecto que puede llegar a lograr. La coherencia se evalúa en 3/5, lo que sugiere que, aunque funcional, el método puede no siempre integrarse bien con el estilo general del juego. La facilidad de integración es relativamente alta (4/5), aunque la facilidad de uso promedio es baja (1/5), probablemente debido a la falta de organización en los archivos y documentación insuficiente.

A pesar de su puntaje, este método puede ser útil para generar pequeñas habitaciones de cueva con un aspecto orgánico. Estas habitaciones pueden luego ser integradas en otros algoritmos que usen objetos prefabricados para construir entornos completos, permitiendo crear cuevas más controladas pero con un aspecto natural. Aunque el desarrollador otorgó una baja puntuación en aspecto, el resto de los encuestados evaluaron el método más favorablemente, lo que sugiere que puede ser adecuado para juegos que se alineen mejor con su estética actual.

Método basado en búsqueda: DFS

El método *DFS* obtiene buenos resultados en adaptabilidad (8/10) y aspecto (5/5), a pesar de esto el cliente expresó su preocupación debido a la repetitividad de los niveles generados, al ser la escena de ejemplo una mazmorra con solo un tipo de habitación (4/5). La facilidad de integración es excelente (5/5), lo que indica que el método se adapta bien al proyecto sin necesidad de una gran cantidad de modificaciones. Sin embargo, la facilidad de uso se ve comprometida (2/5), posiblemente debido a falta de documentación completa y la necesidad de generar assets propios para dar variabilidad al método.

La simplicidad de las habitaciones generadas por este método resulta ventajosa en juegos con una estética y diseño minimalista, donde no se necesite variabilidad en el tamaño de las habitaciones. A pesar de esta simplicidad, el contenido de las habitaciones aún puede variar.

Método basado en búsqueda: Procedural Level Generator

El método *de búsqueda* destaca en todas las categorías, con un puntaje perfecto por parte del desarrollador en adaptabilidad (10/10), aspecto (5/5), coherencia (5/5) y facilidad de integración (5/5). La facilidad de uso también es alta (4/5), aunque podría beneficiarse de mejoras menores, como la inclusión de techos o assets adicionales. Este método es altamente recomendado debido a su facilidad de uso y la calidad del contenido generado.

Además, aunque obtuvo un puntaje casi perfecto, este método presenta el potencial de poder usarse en conjunto con otros como se mencionó en el caso del generador de cuevas con autómatas celulares.

Método basado en fractales: BSP

El método *BSP* ofrece una adaptabilidad decente (8/10) y un buen aspecto (4/5), destacando el interés del cliente para su uso en entornos del tipo arena. Sin embargo, la coherencia (3/5) y la facilidad de integración (2/5) son áreas de preocupación, habiendo encontrado errores con la geometría de las esquinas y la falta de instrucciones claras para agregar colisionadores. La facilidad de uso también es baja (2/5), lo que indica que no todos los parámetros se explicaron de manera clara. Tras arreglar estos aspectos, este método podría ser usado en

juegos con componentes de terror pues estos suelen tener habitaciones opasillos largos cosa que este algoritmo suele generar.

Método basado en Machine Learning: WFC

El método *WFC* presenta puntajes bajos en todas las categorías, incluyendo adaptabilidad (1/10), aspecto (1/5), coherencia (1/5), facilidad de integración (1/5) y facilidad de uso (2/5). Estos resultados sugieren que el método puede no ser adecuado para el juego en cuestión, destacando entre los comentarios que no permite generar niveles de casi ningún tamaño debido a lo pesado del algoritmo y la lentitud del mismo, la falta de coherencia en el resultado generado y la no resolubilidad de los niveles obtenidos. Sin embargo, el cliente mencionó que ve potencial en este para generar habitaciones individuales, lo cual como se mencionó con anterioridad puede ser aprovechado por algoritmos como el DFS o el Procedural Level Generator de la AssetStore para posteriormente generar el entorno completo.

6. Conclusiones

Se logró cumplir con los objetivos planteados para este trabajo, implementando y evaluando al menos un método de cada tipo encontrado en la literatura de generación de contenido procedural. Se encontró que los métodos basados en búsqueda obtuvieron una mejor evaluación y presentaron una mayor flexibilidad al permitir el uso de objetos prefabricados. Esta flexibilidad ofrece un potencial de variabilidad que solo está limitado por los objetos generados para la ejecución de estos algoritmos.

Los principales desafíos de esta investigación residieron en la poca familiaridad con el entorno de Unity así como con el diseño y modelado de objetos prefabricados para los algoritmos que lo requirieran. También destaca la poca documentación existente acerca de PCG para entornos 3D, sobre todo en el área de métodos basados en machine learning, encontrando solo información acerca del Wave Function Collapse [6] como método de este tipo que no requería un dataset de entrenamiento, pues al ser un juego en desarrollo no se cuenta con uno.

Con respecto al trabajo futuro, se plantea la hibridación de algoritmos, como se mencionó a lo largo del documento. Por ejemplo, se podrían utilizar métodos como WFC o el generador de cuevas basado en autómatas celulares para crear habitaciones prefabricadas en lugar de entornos completos, y luego emplear algoritmos de búsqueda para ensamblar estos elementos en el nivel final. Además, se propone explorar distintos métodos de PCG para la decoración de habitaciones, ya que actualmente los niveles generados contienen habitaciones o espacios sin elementos en su interior. Esta tarea también podría abordarse mediante PCG, mejorando así la complejidad y riqueza visual de los niveles.

Referencias

[1] Statista. (2023). *Revenue of the video game industry worldwide from 2012 to 2023*. Recuperado de https://www.statista.com/statistics/1344668/revenue-video-game-worldwide/

[2] SMITH, G. "Understanding procedural content generation: a design-centric analysis of the role of PCG in games" *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*, Association for Computing Machinery, New York, NY, USA, p. 917–926, 2014. https://doi.org/10.1145/2556288.2557341.

[3] Shaker, N., Togelius, J., & Nelson, M. J. *Procedural Content Generation in Games*. Springer, 2016.

[4] Russell, S. J., & Norvig, P. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall, 2010.

[5] ETHERINGTON, T. R., MORGAN, F. J., & O'SULLIVAN, D. "Binary space partitioning generates hierarchical and rectilinear neutral landscape models suitable for human-dominated landscapes." *Landscape Ecology*, v. 37, p. 1761–1769, 2022. https://doi.org/10.1007/s10980-022-01452-6.

[6] Summerville, A., Snodgrass, S., Guzdial, M., Holmgard, C., Hoover, A. K., Isaksen, A., Nealen, A., & Togelius, J. "Procedural Content Generation via Machine Learning (PCGML)." *IEEE Transactions on Games*, 2018.

[7] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018. Disponible en: https://gameaibook.org/book.pdf-

[8] Balint, J. T., & Bidarra, R. "A generalized semantic representation for procedural generation of rooms." En *Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG '19)*, San Luis Obispo, California, USA. Association for Computing Machinery, 2019. https://doi.org/10.1145/3337722.3341848

[9] VIANA, B. M. F., & DOS SANTOS, S. R. "Procedural Dungeon Generation: A Survey." *Journal on Interactive Systems*, Porto Alegre, RS, v. 12, n. 1, p. 83–101, 2021. https://doi.org/10.5753/jis.2021.999. Disponible en: https://sol.sbc.org.br/journals/index.php/jis/article/view/999

[10] "Procedural 2D Dungeon Generation Using Binary Space Partition Algorithm And L-Systems." *2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA)*, October 2023. DOI: https://doi.org/10.1109/IC3INA60834.2023.10285811.

[11] Sebastian Lague. "[Unity] Procedural Cave Generation (E01. Cellular Automata)" YouTube, 2024. Disponible en: https://www.youtube.com/watch?v=v7yyZZjF1z4

[12] Sebastian Lague. *Procedural Cave Generation Repository (Cellular Automata)*. Disponible en: https://github.com/SebLague/Procedural-Cave-Generation

[13] Silverly Bee. *Procedural Dungeon Generator in Unity [TUTORIAL]*. Disponible en: https://www.youtube.com/watch?v=gHU5RQWbmWE

[14] Silverly Bee. *Procedural Dungeon Generation Repository (DFS)*. Disponible en: https://github.com/silverlybee/dungeon-generator

[15] Maxim Gumin *Wave Function Collapse Repository*. Disponible en: https://github.com/mxgmn/WaveFunctionCollapse

[16] Juan Rodriguez *Sitio en la Asset Store del asset usado para el método de búsqueda*. Disponible en: https://assetstore.unity.com/packages/tools/behavior-ai/procedural-level-generator-136626

[17] Sunny Valley Studios. *Unity 3d procedural dungeon generator*. Disponible en: https://www.youtube.com/playlist?list=PLcRSafycjWFfEPbSSjGMNY-goOZTuBPMW

[18] Sunny Valley Studios. *Unity 3d procedural dungeon generator (Binary Space Partitioning)*. Disponible en: https://github.com/SunnyValleyStudio/Unity_Procedural_Dungeon_binary_space_partitioning