



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS

TRABAJO DE FIN DE GRADO

**Generador de niveles de Super Mario:
Una aproximación a la generación procedural de mundos
mediante patrones de diseño.**

Autor:

Fernando Barroso Címbora

Directores:

Jesús Sánchez-Oro Calvo

Juan José Pantrigo Fernández

8 de Mayo de 2020

Resumen

La generación de contenido procedural (GCP) es una potente herramienta dentro de la industria del videojuego, que ha sido aplicada con éxito en multitud de juegos e investigaciones. Esta herramienta, aunque nos proporciona contenido rejugable infinito, muchas veces peca de ser carente de interés, principalmente cuando las creaciones no están regidas por un diseño adecuado.

La generación de contenido procedural, al combinarla con patrones de diseño, podría dar lugar a experiencias mucho más enriquecedoras. En este trabajo discutiremos sobre cómo se pueden combinar estos dos componentes de manera efectiva, creando un generador de niveles de plataformas basado en patrones para apoyar nuestra teoría.

ÍNDICE

1. Introducción.....	5
1.1. Base.....	5
1.2. Generación de contenido procedural	6
1.2.1. En los juegos de plataformas.....	7
1.3. Patrones de diseño	7
1.4. Super Mario y su legado	8
2. Propuesta, objetivos e hipótesis	11
2.1. Propuesta	11
2.2. Objetivos	11
2.3. Hipótesis.....	12
3. Marco teórico.....	13
3.1. Vocabulario.....	13
3.2. Estado del arte.....	14
3.3. Análisis de patrones.....	20
3.3.1. Patrones de diseño de Super Mario	20
3.3.2. Patrones de diseño en juegos de plataformas 2D	23
4. Metodología.....	25
4.1. Herramientas utilizadas: “Mario Editor”	25
4.2. Descripción algorítmica.....	26
4.2.1. Nuestro enfoque: el problema de la mochila.....	26
4.2.2. Descripción de los elementos del nivel	30
4.3. Descripción informática	39
4.3.1. Proceso de generación	39
4.3.2. Implementación algoritmo mochila	40
4.3.3. Estructura del proyecto.....	42
5. Resultados, análisis y discusión.....	45
5.1. Resumen.....	45
5.2. Evaluación del proceso	45
5.2.1. Adaptación al motor	45

5.2.2.	Generación de terreno	46
5.2.3.	Generación de patrones	47
5.2.4.	Integración de patrones en el terreno	48
5.2.5.	Proceso de evaluación	48
5.3.	Evaluación del producto	49
5.3.1.	Evaluación de funcionalidad	49
5.3.2.	Evaluación de diversidad	49
5.3.3.	Evaluación de dificultad	53
6.	Conclusiones y trabajos futuros.....	55
6.1.	Éxito de la propuesta, objetivos e hipótesis.....	55
6.1.1.	Objetivos	55
6.1.2.	Hipótesis.....	55
6.1.3.	Propuesta	56
6.2.	Limitaciones	57
6.2.1.	Re-skin de patrones	57
6.2.2.	Implementación específica	57
6.3.	Contribuciones	57
6.3.1.	Generación aleatoria en juegos de plataformas 2D	57
6.3.2.	Demostración de uso de motor de código abierto para llevar a cabo trabajos de investigación	58
6.4.	En el futuro	58
6.4.1.	Expandir el contenido del generador.....	58
6.4.2.	Generación de niveles on-line	59
6.4.3.	Escalar el modelo a otros juegos de plataformas 2D	59
6.5.	Conclusión	60
7.	Referencias	61

1. Introducción

1.1. Base

Los videojuegos son, como diría Wagner sobre la ópera (Wagner, 1851), la obra de arte total donde se aúnan todas las demás disciplinas artísticas para un fin único. Ciertamente es que pueden tener subpropósitos diferentes que varían desde transmitir cierta idea, despertar una sensación en el jugador, o hasta contar una historia. A pesar de su diversidad, el componente interactivo de los juegos hace que el último cometido que comparten todos ellos sea el de entretener al jugador el máximo tiempo posible.

En la búsqueda de confeccionar la experiencia perfecta de juego, muchos títulos dependen de ofrecer un contenido variado para mantener a los jugadores entretenidos el máximo tiempo posible (Brown, 2018). La repetición y la falta de variedad en un videojuego es causa directa del aburrimiento prematuro por parte del jugador y, por tanto, es una mala práctica que acorta la vida y la calidad del juego en cuestión. Por otra parte, si se abusa de la variedad lo único que se conseguirá es un juego poco cohesionado y un jugador confuso (Dahlskog & Togelius, 2012).

Conseguir la cantidad exacta de innovación es, en consecuencia, un pilar fundamental a la hora de crear niveles. No obstante, la innovación por sí sola no es suficiente para proporcionar una experiencia de calidad. Todo nivel debe estar acompañado de un buen diseño para evitar situaciones indeseables como escenarios frustrantes o injustos, direcciones confusas o retos poco atractivos y/o que se salen del contexto del juego.

En última instancia, un buen diseño se consigue mediante un meticuloso e iterativo método conocido como *playtesting*; proceso mediante el cual se busca identificar cómo los usuarios interactúan con el juego¹, hasta lograr la fórmula perfecta.

Sería pues, interesante, el poder reutilizar el esfuerzo de antiguos trabajos de *playtesting*. Una manera de hacer esto es ser capaz de analizar un diseño de nivel existente, extraer patrones recurrentes del mismo, y generar nuevo contenido a partir de él, de tal manera que la esencia de su contenido y la coherencia se mantienen, a la vez que proporciona nuevas experiencias.

¹ *Playtesting*. (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de <https://es.wikipedia.org/wiki/Playtesting>

1.2. Generación de contenido procedural

La generación de contenido procedural en el mundo de los videojuegos se refiere a la creación, automática o semi-automática, de contenido mediante el uso de algoritmos (Wikipedia, s.f.).

A lo largo de la historia del videojuego, esta técnica se ha ido afinando con los años y ha sido integrada con éxito en multitud de proyectos y en multitud de variantes que van desde la generación de mundo enteros (*No Man's Sky*) y niveles (*Spelunky*) hasta la generación de ítems (*Spore*) y elementos decorativos (*Minecraft*).

Es importante matizar que el término procedural no se refiere (en todos sus usos) a una generación de contenido de manera pseudoaleatoria, sino más bien a todo aquello que haga uso de un proceso de generación automatizada. Convendría decir, incluso, que la aleatoriedad no existe en los videojuegos. Siempre hay reglas básicas. Esto es lo que entendemos por aleatoriedad en forma, no en fondo.

Teniendo esto en cuenta, si queremos utilizar la generación procedural en nuestro juego, debemos en todo momento evitar caer en la aleatoriedad y asegurarnos de que el algoritmo sea capaz de producir una experiencia que tenga sentido para el jugador. De lo contrario, se convierte en un amasijo incoherente de piezas dispersas que rompe con la inmersión, en otras palabras, un mal producto.



Figura 1.1: (a) Un mundo de Minecraft construido a base de bloques puramente aleatorios, en posiciones aleatorias, no tiene mucho sentido...



Figura 1.1: (b) ... pero si le damos cierto orden a esa aleatoriedad, podemos crear mundos infinitos y de calidad

1.2.1. En los juegos de plataformas

A pesar de la versatilidad de la generación procedural, hay algunos géneros que todavía se mantienen relativamente inexplorados por esta herramienta. Uno de ellos es el género de plataformas, nuestro caso de estudio.

La jugabilidad de un nivel de plataformas está fuertemente determinada por las relaciones entre los componentes que lo forman, lo que requiere que dichas relaciones estén fuertemente atadas. Es por esto que automatizar la generación de niveles de plataformas es algo más complicado que generar niveles de otros juegos.

La disposición sin fuertes restricciones de elementos que funciona, por ejemplo, en un generador de mazmorras para un juego RPG, no sería tan efectiva en el caso de ser del género plataformas, pues un cambio minúsculo, como no medir bien la anchura de un abismo, puede transformar un nivel difícil en uno físicamente imposible de superar.

1.3. Patrones de diseño

La concepción del término “patrones de diseño” data de la década de los setenta y de la mano del trabajo del arquitecto Christopher Alexander. Alexander creó un lenguaje de patrones con el objetivo de que, en pocas palabras, cualquier persona pudiera expresar sus ideas y manifestar su capacidad para diseñar, sin necesidad de tener que conocer un lenguaje formal para ello.

“Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, a la vez que ofrece la solución troncal de dicho problema, de tal manera que puedes aplicar la misma solución un millón de veces, y en todas haciéndolo de una manera diferente”- (Alexander, Silverstein, & Ishikawa, 1977).

Un patrón tiene, en esencia, dos componentes: problema y solución. El valor de los patrones no recae, sin embargo, en la solución en sí, sino en la generalización y abstracción de la misma para crear un espectro de posibles soluciones.

Esta manera de pensar llegó al diseño de software en 1994, cuando los ingenieros Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides publicaron una serie de soluciones a los problemas más comunes y recurrentes en el diseño orientado a objetos (Gamma, Helm, Johnson, & Vlissides, 1994). Las soluciones que plantearon no eran un diseño específico que hubiera que cumplir estrictamente para solucionar el problema, sino más bien unas plantillas

que podían ser usadas en multitud de situaciones diferentes. Ejemplos de estas soluciones son a lo que hoy nos referimos como “Herencia” o “Polimorfismo”.

En los videojuegos los patrones de diseño están presentes por doquier, y, en la misma línea del diseño de software, se interpretan como una manera de dar respuestas a problemas. Esta vez, problemas puestos por el diseñador de niveles o simplemente creados por las propias reglas del juego.

Cuando nacen buenos patrones, estos se repiten con recurrencia en la industria. Ejemplo de ello es la colección de monedas en *Super Mario*, juego que introdujo por primera vez un sistema de recompensa tras recolectar cierto número de elementos, y que ha sido adoptado por *Sonic the Hedgehog* (1991) usando anillos como elemento coleccionable con el mismo efecto, y con frutas en el caso de *Crash Bandicoot* (1996), entre muchos otros.

Vale la pena resaltar que, por muy útiles que sean, la ausencia de patrones no implica automáticamente un mal diseño. Al fin y al cabo, son guías, y es decisión del diseñador si usar tales conceptos o no.

1.4. Super Mario y su legado

El hecho de que hayamos escogido el videojuego Super Mario como caso de estudio no es casualidad. A pesar de que con los años se hayan explorado nuevos géneros dentro de la saga como *Super Mario Galaxy* (2007) o *Paper Mario* (2000), nos centraremos en su serie principal, el género de plataformas 2D, que comenzó su andadura con el aclamado *Super Mario Bros* (1985).

Este juego fue un hito tanto por su papel en la industria del videojuego como por la trascendencia de su diseño. Tanto es así, que los mismos patrones de diseño que se usaron en *Super Mario Bros* se llevan repitiendo en sus juegos ya por más de 30 años. Así lo determinaron Steve Dahlskog y Julian Togelius en su análisis de más de 40 niveles de Super Mario a lo largo de 30 años de lanzamientos (Dahlskog & Togelius, 2012). ¿Significa esto que nos han estado vendiendo el mismo juego todo este tiempo? Nada más lejos de la realidad. De hecho, la manera que tienen de añadir frescura y novedad manteniendo un diseño impecable con cada entrega ha sido estudiada (Thompson, 2015) y se podría resumir en tres puntos:

1. Con el aumento de la capacidad de las consolas a lo largo de los años, también ha aumentado la densidad de los niveles (o número de patrones por sección). De esta manera, se da pie a muchas más formas de explorar las soluciones que plantean.
2. Con cada nuevo juego, los patrones de juegos antiguos se descartan o se les cambia totalmente la apariencia
3. Los patrones nuevos que surgen con cada entrega se inspiran siempre en patrones previos, o en la mezcla de varios

La lectura que debe hacerse de esto nos lleva a recalcar el papel crucial de los patrones en la industria, y cómo pueden prevalecer sin problemas a lo largo del tiempo para seguir proporcionando experiencias nuevas y de calidad.

2. Propuesta, objetivos e hipótesis

2.1. Propuesta

A pesar de que otros géneros han integrado con eficacia la generación procedural de contenido en sus juegos para extender el *gameplay* y aportar rejugabilidad, hemos visto que la propia naturaleza de los juegos de plataformas hace que su representación en el mundo de la automatización de contenido todavía sea muy escasa.

Sería conveniente, pues, automatizar la construcción de niveles de plataformas 2D por medio de patrones de diseño (problema-solución), para poder generar contenido sin la preocupación de estar comprobando constantemente si se ha generado algún obstáculo imposible de superar para el jugador.

La propuesta de este proyecto es, por tanto:

Usando a los juegos de Super Mario como modelo, evaluar la utilidad de los patrones de diseño cuando se usan en la generación automática de niveles de plataformas 2D en dos aspectos fundamentalmente: (i) la complejidad que supone su proceso de desarrollo y (ii) la efectividad de los niveles obtenidos.

2.2. Objetivos

Para poder lograr esta propuesta, se llevarán a cabo los siguientes objetivos:

1. Llevar a cabo un análisis de técnicas de generación procedural de niveles de Super Mario pre-existentes, resaltando los pros y contras de cada uno.
2. Identificar y analizar patrones de diseño, tanto específicos de la saga Super Mario, como compartidos entre múltiples títulos de plataformas 2D.
3. Programar un generador de niveles de Super Mario usando patrones, y documentando el proceso de desarrollo simultáneamente.
4. Evaluar la complejidad del proceso de desarrollo, con la intención de determinar si los patrones de diseño son una solución práctica para la generación de niveles de plataformas 2D.
5. Evaluar la efectividad del resultado basándonos en la diversidad y en la funcionalidad de los niveles producidos.

2.3. Hipótesis

Basándonos en el desglose de los objetivos anteriores, podemos establecer la hipótesis como la siguiente:

Aplicar patrones de diseño a la generación procedural de niveles de Super Mario en 2D proporcionará una serie de niveles diversos y funcionales, y establecerá un modelo que podrá ser escalado con éxito a otros títulos de plataformas 2D para su generación automática.

Esta hipótesis se centra más en el resultado que en el desarrollo como tal. Para poder determinar si los niveles son diversos y funcionales se documentará información relevante del proceso de desarrollo en este escrito. Más tarde se evaluarán dichas aptitudes realizando un gran número de pruebas sobre el generador y, tras analizar los resultados obtenidos, podremos concluir si el modelo de generación por patrones es válido para otros juegos.

3. Marco teórico

3.1. Vocabulario

Dada la cantidad de extranjerismos presentes en el mundo del videojuego, y la imprecisión con la que se refieren a ellos algunos términos en español, es importante dedicar unas líneas antes de profundizar más para conocer el vocabulario que usaremos repetidamente en este proyecto. La lista de a continuación está extraída de diversas fuentes, que incluyen los trabajos de Clark y Anthropy (Clark & Anthropy, 2014), Bjork y Holopainen (Bjork & Holopainen, 2005) y la wiki de patrones de diseño (Bork, s.f.).

Tabla 3.1: Vocabulario recurrente del proyecto

Nombre	Definición
Avatar	La entidad que el jugador maneja dentro del juego. Por ejemplo: Mario.
Amenaza	Cualquier entidad que dificulta al jugador cumplir su objetivo. Por ejemplo: Pinchos
Asset	
Coleccionable	Un objeto dentro del juego que puede ser recolectado por los jugadores. Por ejemplo: Monedas
<i>Checkpoint</i>	Zona de guardado automática en un nivel
Enemigo	Una amenaza que toma la apariencia de un personaje. Por ejemplo: Goomba.
Escena	Una sección del nivel que presenta un concepto o reto.
<i>Gameplay</i>	Conjunto de acciones que puede realizar un jugador para interactuar con juego o la forma en la que este interactúa con el propio jugador
<i>Grid</i>	En los videojuegos, se refiere a un nivel en el que los elementos se disponen en una cuadrícula invisible
Mecánica	Cualquier elemento en un juego, visible o no, que refuerza una regla. Por ejemplo: los puntos de control hacen que el jugador no tenga que empezar desde el

	principio del nivel si muere después de haber pasado por ellos
Nivel	Es una sección del juego donde los jugadores tienen que cumplir un objetivo. En el caso de Mario, llegar al final del mismo.
Objeto	Cualquier entidad que aparezca en una escena y que puede cambiar de estado. Dentro de los objetos están comprendidos los enemigos, power-ups, las amenazas...
<i>Power-up</i>	Un coleccionable que tiene un impacto positivo en el jugador al ser recogido. Por ejemplo: el champiñón verde aporta una vida extra a Mario.
Regla	Una regla describe cómo se juega y se controla el juego en cuestión.
<i>Tile</i>	Cualquier entidad que forma parte de la estructura de un nivel. Estos pueden ser sólidos (si no pueden ser atravesados) o decorativos (no interactivos).
<i>Tilemap</i>	Archivo de imagen en el que se agrupan todos los tiles que se usan en un nivel. A la hora de dibujar un tile, se renderiza la sección correspondiente de la imagen que contiene ese tile.

3.2. Estado del arte

Hemos hablado en la sección 1.4 de la trascendencia de Super Mario y de cómo lleva siendo un icono de los videojuegos durante varias décadas. Sin embargo, recientemente, el fontanero ha encontrado un nuevo propósito más allá del puro entretenimiento. Pues, para algunos, es también un campo de pruebas a favor de la investigación de la inteligencia artificial. Tanto es así, que incluso se celebró una competición anual, desde 2009 a 2012, de evaluación de sistemas de inteligencia artificial usando a Super Mario como base: *the Mario AI competition* (Togelius, Karakovskiy, & Baumgarten, 2010)

Lo que hace que Super Mario sea un atractivo objeto de estudio y no otros juegos de plataformas que siguieron sus pasos con éxito (*Sonic, Contra, Metroid...*) es, después de todo, su popularidad e influencia. No obstante, algo que también hace que los juegos de Mario fascinen

a los investigadores es el balance casi perfecta entre simplicidad y complejidad que ofrece su diseño. En otras palabras, las mecánicas son lo suficientemente complejas para ofrecer retos interesantes a los algoritmos, pero no llegan al punto de ser imposibles de resolver. Aún más allá de las mecánicas, la propia estructura de los niveles también ejemplifica un punto medio entre simplicidad y complejidad. “Los niveles de Super Mario muestran patrones marcados, pero también son en parte abstractos”, escribieron Guzdial y Riedl (Guzdial & Riedl, 2016). Puede que los niveles parezcan simples a simple vista, pero su diseño tiene la dificultad justa para ofrecer un gran potencial en la investigación de sistemas de inteligencia artificial.

Procedemos a continuación a ver las tres herramientas que son, a fecha del proyecto, las más recientes y conocidas en la generación de niveles procedurales de Super Mario.

Generación por redes neuronales

Empezaremos por hablar del trabajo de Adam Summerville (Summerville & Mateas, 2016). Este autor ha llevado a cabo dos aproximaciones a la generación procedural de niveles de Super Mario utilizando aprendizaje automático, pero nos centraremos en la segunda por ser la más reciente y, en esencia, una mejora de la primera.

En esta segunda aproximación, se propone una variante de red neuronal artificial conocida como red LSTM (de *long short-term memory network*, en inglés). Las redes LSTM [referencia] datan de la década de los 90 y son muy buenas para manejar secuencias de datos, dado que tienen una memoria asociada a la información entrante, permitiendo no sólo leer información nueva, sino dictar cuándo recordar y cuando olvidar datos de iteraciones pasadas.

En este trabajo, se probaron con diferentes configuraciones de redes neuronales hasta dar con una capaz de leer niveles y generar el suyo propio. Una vez con la red adecuada, y tras haber definido qué es sólido, rompible, enemigos, tuberías, monedas, bloques y demás componentes del juego, esta LSTM se somete a un proceso de entrenamiento con 15 niveles del *Super Mario Bros* original.

Tras entrenar a la red neuronal con un cierto nivel de confianza, es cuando se le pide que genere nuevos niveles a partir del entrenamiento recibido, como podemos ver en el ejemplo de la Figura 3.1.

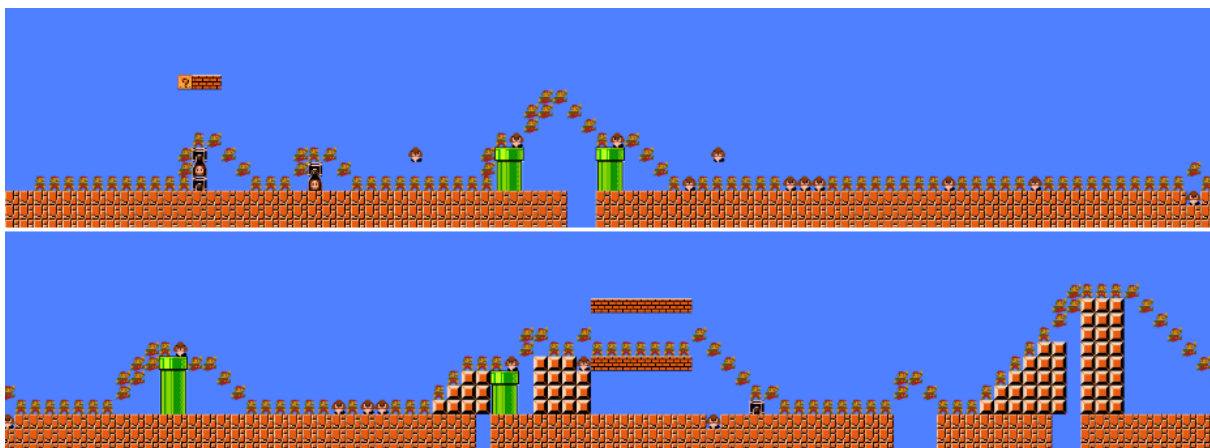


Figura 3.1: Ejemplo de nivel generado por la red neuronal (Créditos: Adam Summerville)

Una vez generados, los niveles se someten a una serie de evaluaciones. Estas evaluaciones — en palabras del autor — no están dirigidas a medir la credibilidad que ofrece un nivel generado en términos de hacerse pasar por uno de un juego original de Mario. Es decir, no se busca replicar las estructuras con fidelidad. Por el contrario, lo que se pretende es crear niveles que, a la vez que compartan propiedades similares con los originales, añadan componentes propios y novedosos, resultado del aprendizaje de la red neuronal.

Estas evaluaciones no sólo determinan si el nivel se puede completar o no (si hay obstáculos imposibles) mediante un *bot* que lo recorre, sino que también miden el porcentaje de espacio vacío — el espacio por el que el jugador puede moverse — del nivel, el mínimo número de saltos que debe hacer Mario para pasarse el nivel y otras métricas relativas a la dificultad del nivel o a la linealidad del mismo. Con toda esta información obtenida se retroalimenta la red neuronal y se consiguen mejores niveles con cada iteración.

Las redes neuronales son un método de aprendizaje automático muy útil, y de la forma en las que se utilizan en este trabajo, junto con las evaluaciones de funcionalidad, son un candidato con mucho potencial para generar niveles interesantes de jugar.

No obstante, el inconveniente es que lo que tiene de útil también lo tiene de complejo. El tiempo de aprendizaje de una red neuronal crece exponencialmente conforme se le suman variables a tener en cuenta, y en este caso, en el que estamos considerando la construcción de niveles enteros, se trata de un proceso bastante largo. Si además se llevan a cabo las evaluaciones mencionadas el proceso se alarga bastante más, pues sólo el adiestramiento del bot que recorre el nivel para detectar si es jugable lleva horas, si no días, de aprendizaje (Portoff, 2015).

Generación por vídeo

El segundo trabajo del que hablaremos es el de Matthew Guzdial (Guzdial & Riedl, 2016). Para generar niveles, el objetivo de Guzdial no sólo era el de capturar información de niveles, sino también información de cómo los jugadores interactúan en dichos niveles. Para ello, aprende sobre los niveles de Super Mario viendo a personas jugar en YouTube.

Como se detalla en la memoria del autor, el proyecto usa OpenCV ² — una biblioteca de *software* de código libre para visión por computador — para procesar cada frame de un video dado. A partir de esta premisa, los objetivos del proyecto se dividen en: 1) determinar hasta qué punto se puede aprender sobre la disposición de niveles a través un archivo de video y 2) cómo se puede representar el conocimiento sobre diseño, oculto a simple vista y representado implícitamente en el gameplay grabado en vídeo, en la generación de niveles

El resultado es, como podemos apreciar en la Figura 3.2, que para ser un sistema que aprende únicamente mediante archivos de vídeo, los niveles resultantes son asombrosamente precisos.

Este método de generación resulta increíblemente novedoso, y como se ha comentado, da pie a una extensa área de investigación en la que aplicar y perfeccionar el aprendizaje por video. Sin embargo, resulta algo limitante en el sentido de que se necesita reunir un gran número de videos para recrear niveles con fidelidad. Si le sumamos que los niveles que aparezcan en los videos deben tener los mismos *sprites* para que el sistema los reconozca, se acota más la búsqueda. Además de esto, en palabras de Matthew, para prevenir la muerte de Mario — que causa una pantalla

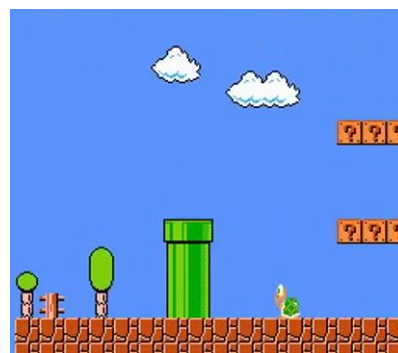


Figura 3.2: Fragmento de nivel generado mediante datos extraídos de vídeos (Créditos: Matthew Guzdial)

en negro — interrumpa el proceso de análisis, sólo se usan fragmentos de video donde los jugadores no mueren. Tener que recortar y procesar los videos hasta dar con el fragmento adecuado no hace más que dificultar aún más la recogida de datos.

² OpenCV (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de <https://es.wikipedia.org/wiki/OpenCV>

Generación por patrones de diseño

Por último, hablaremos del trabajo que más relación guarda con el nuestro, el llevado a cabo por Steve Dahlskog (Dahlskog & Togelius, 2012), que también se centra en el estudio de los patrones de diseño para abordar la generación procedural. Sin embargo, en su trabajo estos se abordan de una manera diferente, como veremos a continuación.

La investigación de Steve comienza identificando 23 patrones de diseño, exclusivos de Super Mario, recogidos a lo largo de 20 niveles del primer juego. Con los patrones identificados, los niveles de los que se recogieron se analizan por separado: se descomponen en 200 fragmentos verticales con la intención de examinar cómo dichos fragmentos se conectan con otros y determinar qué patrones existen en esas conexiones de fragmentos.

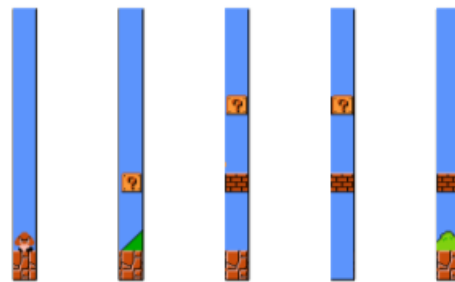


Figura 3.3: Fragmentos verticales de un nivel

Finalmente, usando una colección de los fragmentos verticales encontrados con más frecuencia en los niveles analizados, y teniendo en cuenta las relaciones entre ellos, el sistema puede construir un Modelo de Markov ³ en base a ellos e intentar reproducir niveles que tengan subsecuencias de fragmentos similares a los niveles originales. Un Modelo de Markov, a diferencia de las redes LSTM que ya hemos visto, se caracteriza por su falta de memoria, dando lugar a que la probabilidad de que ocurra un evento dependa solamente del evento inmediatamente anterior.

Los niveles resultantes no sólo reflejan patrones específicos de juegos de Mario existentes, sino que los concatena en formas nunca vistas antes. Con esto se mantiene la fórmula original de los juegos mientras se le añade frescura con nuevas estructuras.

³ Cadena de Márkov (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de https://es.wikipedia.org/wiki/Cadena_de_M%C3%A1rkov

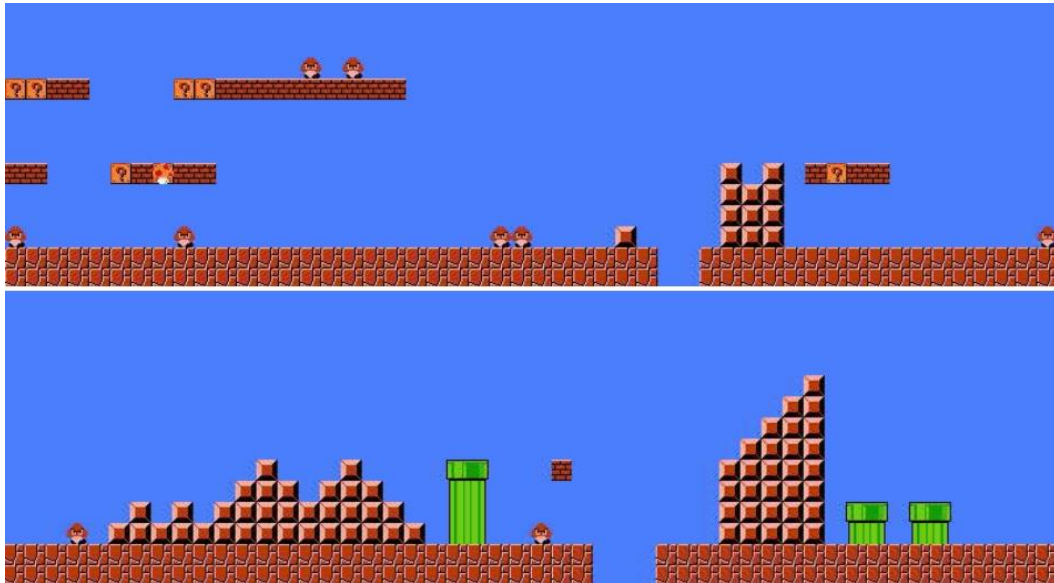


Figura 3.4: Nivel ejemplo resultado del generador (Creditos: Steve Dahlskog)

El análisis de patrones de Steve resulta increíblemente útil a la hora de entender la lógica y estructura de los niveles del fontanero, y ha supuesto un antes y un después en la manera de entender qué factores hacen de un juego de plataformas, un juego de Super Mario.

Sin embargo, un inconveniente de su implementación que llega a ser problemático es que no hay una garantía absoluta de que los niveles generados sean 100% jugables. La estructura de los nuevos patrones a generar no está supervisada, y está a merced del conjunto de probabilidades que maneja el Modelo de Markov que se produzca algo que tenga sentido. Por otra parte está el tema de que, mediante el uso de patrones, el sistema busque generar niveles que se parezcan a los originales. Aunque vemos que lo consigue con efectividad si nos fijamos en los patrones uno por uno, la sensación de conjunto de todos ellos en nivel no lo logra del todo. Esto es porque no existen parámetros que indiquen qué patrones resultan más retantes que otros y no se tiene en cuenta que se ha dibujado cierta zona del nivel cuando ya la hemos “pintado”, debido al sistema sin memoria de Markov. Por todo esto, hay nula sensación de progresión en el nivel, llegando a parecer más un escaparate de patrones colocados en un lienzo que un nivel propiamente dicho.

Nuestra propuesta

Aunque se desarrollará con más profundidad más adelante, la principal diferencia nuestra entre propuesta y los trabajos expuestos es que no estamos usando elementos únicamente del diseño de niveles de Super Mario, estamos llevando el concepto de patrón de diseño a su máximo exponente y convirtiendo abstracciones como Guía y Forma (sección 3.3.2) en elementos

parametrizables. Ya que a pesar de que las herramientas de generación expuestas se hayan enfocado en Super Mario, en última instancia el uso de Super Mario como caso de estudio en nuestro proyecto no es más que el punto de partida con el que elaborar un sistema de generación flexible y sentar las bases de un modelo escalable a otros juegos de plataformas.

Otro factor importante del que nos diferenciamos de los trabajos existentes es que nuestro enfoque pretende darle más control al diseñador de los niveles que se crean. En otras palabras, a diferencia de los trabajos previos, no dejamos a merced de la máquina que aprenda y defina la lógica de decisiones de diseño que afecten, por ejemplo, a la manera en la que deberían agruparse los *tiles*, qué *tiles* se usan en qué contexto o qué texturas deberían aplicarse a los *tiles* y en qué situaciones.

Lo que buscamos es diseccionar todo ese conocimiento sobre diseño de juegos ya existente y a nuestra disposición, y usarlo a nuestro favor para parametrizar el generador en base a ello. Con esto se consigue “lo mejor de los dos mundos”, todo lo bueno que aporta el conocimiento sobre diseño de juegos recopilado a lo largo de las décadas por el ser humano, con la rapidez que ofrece la generación automática de niveles.

3.3. Análisis de patrones

Antes de empezar a confeccionar nuestro proyecto es necesario identificar una batería de patrones a partir de los cuales basar la generación aleatoria de niveles. Estos patrones los dividiremos, según de dónde los extraigamos, en patrones recurrentes en juegos de Mario, y en patrones generales encontrados en múltiples títulos de plataformas 2D. A efectos prácticos, ambos tipos cumplen con la definición de patrón de diseño expuesta anteriormente, sin embargo, cuentan con matices que los hacen diferentes. Los patrones de Super Mario, al ser únicos de esta serie de juegos, se caracterizan por ser específicos y contar con una estructura definida, mientras que los generales se basan en ideas o conceptos más que en algo tangible.

3.3.1. Patrones de diseño de Super Mario

En primer lugar, comenzaremos por analizar patrones que se repiten en títulos de Super Mario en 2D, acompañados por algunas imágenes para facilitar su comprensión. Estos patrones comprenden la disposición de los enemigos y la agrupación de obstáculos del entorno.

Tabla 3.2: Patrones de Super Mario Bros agrupados por tema

Enemigos	
Enemigo	Un enemigo suelto
Horda de 2	Dos enemigos dispuestos seguidamente
Horda de 3	Tres enemigos dispuestos seguidamente
Horda de 4	Cuatro enemigos dispuestos seguidamente

Hoyos	
Hoyo	Un hoyo individual que supone una derrota si el avatar cae en él.
Hoyos múltiples	Varios hoyos dispuestos seguidamente.
Hoyos variables	Varios hoyos dispuestos seguidamente y de anchura variable.
Hoyo(s) con enemigo	Hay un enemigo volador en mitad del hoyo.
Hoyo con pilar	En el suelo entre hoyo y hoyo hay tuberías de distinta altura
Hoyo con pilar y pirañas	Misma estructura que el patrón anterior, con la diferencia de que ahora hay plantas piraña en las tuberías (un tipo de enemigo que sale y se esconde de manera intermitente en las tuberías)

Valles	
Valle	Dos tuberías dispuestas verticalmente de forma paralela, que forman un valle entre ellas
Valle con enemigos	Entre esas dos tuberías hay uno o varios enemigos
Valle con pirañas	De las tuberías salen plantas piraña
Valle con enemigos y pirañas	Una combinación de los dos patrones anteriores

Escaleras	
Escalera ascendente	Una serie de bloques dispuestos en forma de escalera ascendente
Escalera descendente	Una serie de bloques dispuestos en forma de escalera descendente
Escalera valle	Una escalera ascendente y una descendente dispuestas en serie
Escalera valle con hoyo	Entre las dos escaleras en serie hay un hoyo
Escalera valle con enemigo	Entre las dos escaleras hay uno o varios enemigos



Figura 3.5: (a) Patrón Horda de 2 Enemigos



Figura 3.5: (b) Patrón Hoyo con Pilar y Pirañas



Figura 3.5: (c) Patrón Valle

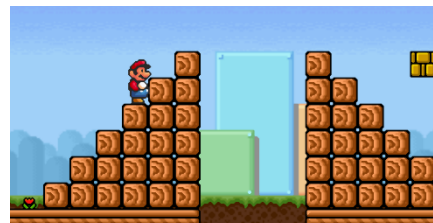


Figura 3.5: (d) Patrón Escalera valle con hoyo

A simple vista, parecería que los patrones expuestos son sólo variaciones de cuatro patrones troncales. Sin embargo, la razón por la que hemos hilado fino y elegido un análisis tan detallado es justo para recalcar que existen diferencias significativas, en términos de interactividad, entre patrones que estructuralmente son muy parecidos.

Pongamos un ejemplo, entre los patrones de enemigos. No es lo mismo enfrentarse a una horda de cuatro que a dos hordas de dos. En los dos patrones hay cuatro enemigos, pero la altura del salto y la carrerilla que tendrá que coger Mario variará en cada caso.

Si a esto le sumamos que se pueden combinar patrones, las posibilidades se multiplican. Un patrón compuesto consiste en dos componentes colocados tan cerca uno de otro, que para resolverlos se requiere una acción diferente, o coordinada, que no se requeriría en el caso de enfrentarse a esos patrones individualmente. Esto requiere que el jugador tenga conocimiento de los dos problemas por separado para poder sintetizar una solución combinada y resolverlo con eficacia.

3.3.2. Patrones de diseño en juegos de plataformas 2D

A continuación, se exponen patrones generales encontrados recurrentemente en múltiples títulos de plataformas 2D. Es importante aclarar que existen muchos más patrones de los expuestos en esta sección, sin embargo, no todos los patrones existentes son aplicables a todos los juegos o niveles. Por esta razón, la elección se ha basado en los patrones más comunes y con mayor impacto en el jugador, así como los que tienen más cabida en el tipo de juego sobre el que vamos a trabajar. En este caso, un juego de Super Mario.

- **Guía**

Cuando se juega un nivel, es posible que los jugadores pierdan la noción de a dónde dirigirse, especialmente cuando la exploración es un aspecto importante en el diseño.

El patrón denominado “Guía” hace referencia al uso de elementos no verbales para guiar al jugador en una dirección determinada. El ejemplo más recurrente de la guía es el de orientar al jugador hacia el final del nivel, pero también puede usarse para sugerirle zonas secretas o especiales ocultas a simple vista.

Una manera sencilla de guiar al jugador por el nivel es a través de la forma y composición del mismo. Otra de las maneras más comunes de aplicar este patrón es mediante la disposición de los elementos coleccionables (p.ej: monedas en *Super Mario*, bananas en *Donkey Kong*, etc).

En todas sus variantes, la intención principal de este patrón será la de llevar la mirada del jugador a la dirección deseada.

- **Múltiples caminos o *Branching***

El término *Branching* se refiere al patrón mediante el que se ofrecen al jugador varias vías para alcanzar su objetivo.

El hecho de darle a los jugadores la posibilidad de elegir su camino les proporciona una sensación de empoderamiento y de mayor control sobre sus actos.

- **Riesgo y recompensa**

Cuando se presentan varias rutas posibles, una de ellas puede ser una alternativa segura con una recompensa pequeña, mientras que la otra requiera un poco más de habilidad, pero ofrezca una recompensa mayor. Esta conducta es conocida como patrón riesgo-recompensa, y es muy útil para dar un reto extra a jugadores más experimentados, sin comprometer la dificultad global del nivel.

4. Metodología

4.1. Herramientas utilizadas: “Mario Editor”

Para el desarrollo de este proyecto se ha optado por usar “Mario Editor”. “Mario Editor” es un herramienta de creación de niveles de Super Mario desarrollado por *Hello Fangaming*. (Hello Fangaming, s.f.)

Muy al estilo de “Super Mario Maker” (*Nintendo*, 2015), “Mario Editor” ofrece una interfaz que permite al usuario crear niveles desde cero mediante el posicionado manual de elementos en un “lienzo” cuadriculado (Figura 4.2). Todos los elementos de los juegos, así como su funcionalidad, está autocontenida en la herramienta: movimiento de Mario, comportamiento de enemigos, colisiones de paredes, etc; de manera que no es necesario saber programar para ensamblar un nivel.

Una vez guardados, los niveles se almacenan como archivos JSON que contienen toda la información del mismo. En vez de usar la interfaz de “Mario Editor” para crear un nivel manualmente, se creará un código capaz de generar niveles automáticamente, de manera prodecural y aplicando patrones de diseño en su generación.

Usaremos el lenguaje C++⁴ para crear el código de generación, y una vez ejecutado y obtenido el nivel, la información del mismo será estructurada en un archivo *.JSON*⁵ para poder ser compilado y ejecutado por “Mario Editor”. Se ha elegido C++ como lenguaje de programación por su portabilidad (poder compilar nuestro código en diferentes plataformas), por su eficiencia y robustez, y por ser el lenguaje por excelencia más utilizado en la creación de videojuegos (García, 2016).

JS eval

```
{
  "MAIN": {
    "MUSIC": "LAND",
    "OBJECTS": {
      "1,18": "MARIO"
    },
    "CLASSIC_SCROLL": 0,
    "DAREDEVIL": 0,
    "TILES_BG": {
    },
    "WIDTH": 192,
    "MODIFIERS": {
    },
    "HEIGHT": 27,
    "BACKGROUND": "HOUSE",
    "MARKERS": {
    },
    "KEEP_POWERUP": 0,
    "RAIN": 0,
    "TILES": {
      "4,20": "1,12",
      "1,20": "1,12",
      "2,20": "1,12",
      "3,20": "1,12"
    },
    "SNOW": 0
  }
}
```

Figura 4.1: Archivo JSON de nivel de prueba

⁴ C++ (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de <https://es.wikipedia.org/wiki/C%2B%2B>

⁵ JSON (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de <https://es.wikipedia.org/wiki/JSON>

Por otra parte, la Figura 4.1 muestra el archivo .JSON que almacena un nivel de prueba. En él se puede apreciar cómo se define la estructura del nivel mediante etiquetas: desde la anchura total del mapa, qué tipo de tile va en qué posición o hasta la música que suena. Será añadiendo etiquetas y modificando el valor de las mismas con valores generados mediante los algoritmos procedurales, que seremos capaces de crear nuestros niveles y ejecutarlos en Mario Editor.

Con este código no sólo conseguimos múltiples niveles de Super Mario en tan solo unos segundos, sino que todos ellos serán diferentes entre sí y ofrecerán retos variados e interesantes mediante el uso de los patrones de diseño.

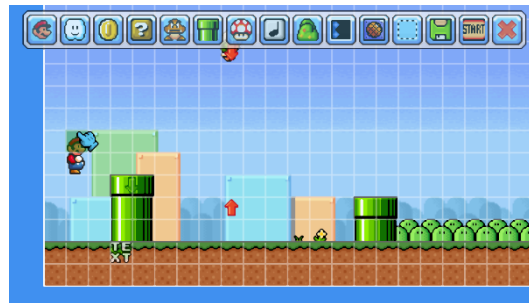


Figura 4.2: Interfaz cuadriculada de Mario Editor

4.2. Descripción algorítmica

4.2.1. Nuestro enfoque: el problema de la mochila

Los niveles a generar se plantean como un lienzo sobre el que colocar patrones de la manera más efectiva, mientras se pone un gran énfasis en mantener lo que llamamos las tres C: coherencia, cohesión y continuidad del nivel.

Gran parte de las decisiones de diseño tomadas durante el desarrollo sobre cómo colocar los patrones de la manera más efectiva se apoyan en **el problema de la mochila**, un problema de optimización combinatoria que busca la mejor solución entre un conjunto finito de posibles soluciones a un problema.

Como su nombre indica, el algoritmo modela una situación análoga al llenar una mochila, incapaz de soportar más de un peso determinado, con todo o parte de un conjunto de elementos, cada uno con un peso y valor específico (Wikipedia, s.f.) . En nuestro caso, los niveles serán nuestra mochila, mientras que los elementos a elegir serán los patrones de diseño. Estos tendrán

un peso asociado cuyo valor dependerá de la dificultad de pasar a través de él y/o de las amenazas que presenten al jugador.

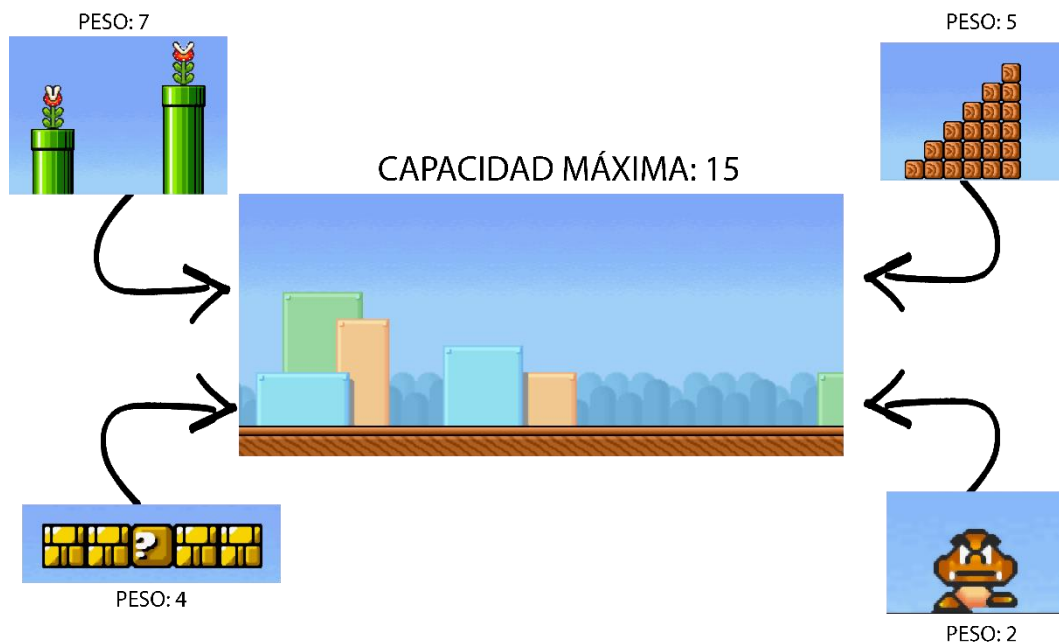


Figura 4.3: Mochila vacía (nivel) con posibles objetos (elementos) a su alrededor

Supongamos que nuestro nivel tiene una capacidad máxima de 15 unidades. El nivel vacío (imagen del centro) es la mochila y los elementos que lo rodean los posibles patrones a colocar en ella. Dichos elementos se irán añadiendo a la mochila progresivamente mientras la suma del peso total de todos ellos sea menor que la capacidad máxima del nivel. Cuando la mochila está llena, se ordenan sus elementos y se disponen en el nivel de menor a mayor peso. Así se evita que un patrón más “fácil” de superar aparezca después de uno más “difícil”, rompiendo así con la curva de dificultad.

Por motivos de regulación de dificultad en un nivel, en el proyecto se usará una implementación personal del problema de la mochila. De carácter semi-aleatorio, los elementos para rellenar la mochila no se eligen por su valor, sino que se rigen bajo unos porcentajes que indican la probabilidad de ser elegidos. Dichos porcentajes cambian dependiendo de la dificultad inicial elegida, dando lugar, por ejemplo, a mayores patrones de peso alto en un nivel “DIFÍCIL” y mayores patrones de peso bajo en un nivel “FÁCIL”.

El motivo de esta implementación personal mediante porcentajes, y no siguiendo el modelo tradicional de la mochila (mediante la asignación de valores a los elementos) se debe a que no queremos maximizar el número de patrones de cierto peso. De hacer esto, todos los niveles de

una cierta dificultad serían prácticamente iguales, pues al intentar maximizar se terminarían eligiendo los mismos patrones. Añadiendo el componente de los porcentajes, se le da prioridad a los del peso que dicta la dificultad, pero no se impide que en determinadas ocasiones se elijan otros patrones. Con esto no solo conseguimos niveles más variados, sino una mejor progresión, ritmo y sensación de dificultad ascendente en los mismos. Por ejemplo, en un nivel de tipo “DIFÍCIL” tener patrones difíciles desde el principio resulta algo frustrante, pero añadiéndole esos patrones de peso bajo al principio se le da al jugador la oportunidad de conocer las mecánicas y de calentar antes de las pruebas de mayor peso. Lo mismo pasa con el de tipo “FÁCIL”, tener todo lleno de patrones fáciles no aporta ningún reto y no se la da al jugador la oportunidad de poner a prueba lo aprendido con retos más difíciles.

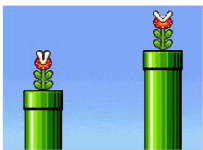

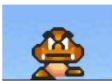


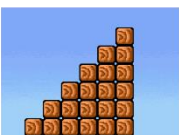






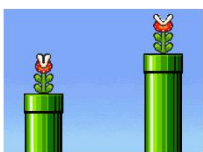
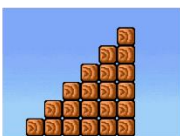


M O C H I L A	ELEMENTOS
<div data-bbox="292 1003 359 1070">①</div> <div data-bbox="392 965 595 1113">  </div> <div data-bbox="469 1137 799 1171">ESPACIO DISPONIBLE: 8</div>	<div data-bbox="1078 929 1260 1061">  </div> <div data-bbox="1275 1014 1386 1093">  </div> <div data-bbox="1070 1120 1267 1176">  </div>
<div data-bbox="292 1249 359 1317">②</div> <div data-bbox="387 1211 590 1359">  </div> <div data-bbox="616 1218 794 1352">  </div> <div data-bbox="469 1368 799 1402">ESPACIO DISPONIBLE: 3</div>	<div data-bbox="1275 1234 1386 1312">  </div> <div data-bbox="1070 1328 1267 1384">  </div>
<div data-bbox="292 1473 359 1541">③</div> <div data-bbox="384 1435 587 1576">  </div> <div data-bbox="612 1440 791 1574">  </div> <div data-bbox="825 1473 1019 1529">  </div> <div data-bbox="469 1588 807 1621">ESPACIO DISPONIBLE: -1</div>	<div data-bbox="1275 1458 1386 1536">  </div>
<div data-bbox="292 1704 359 1771">④</div> <div data-bbox="384 1666 587 1816">  </div> <div data-bbox="612 1675 791 1809">  </div> <div data-bbox="860 1704 971 1783">  </div> <div data-bbox="469 1832 799 1865">ESPACIO DISPONIBLE: 1</div>	<div data-bbox="1070 1733 1267 1789">  </div>

Figura 4.4: Diagrama explicativo del funcionamiento del algoritmo de la mochila en nuestro proyecto

La Figura 4.4 muestra un caso de ejemplo del algoritmo para un tipo de nivel “DIFÍCIL”. Dada la dificultad seleccionada, la mochila comienza a elegir patrones, priorizando los de peso alto. Sin embargo, cuando se intenta insertar el patrón de *Bloques* (el de peso 4) se sobrepasa el espacio disponible, así que se devuelve al conjunto de elementos y se elige otro patrón. Cuando no se pueden añadir más elementos sin que se sobrepase el espacio, finaliza el algoritmo.

Una vez llena la mochila, se ordenan los patrones de menor a mayor peso y se disponen el nivel, dando como resultado la Figura 4.5.



Figura 4.5: Patrones ya dispuestos en el nivel

Este diagrama sirve para ilustrar de una manera sencilla el concepto de cómo usamos el problema de la mochila en nuestro proyecto. No obstante, en la realidad, los patrones de misma índole se agrupan en mochilas diferentes, de manera que, por ejemplo, existe una capacidad máxima para los enemigos y otra distinta para los bloques. Esto se hace para que el peso de un *Goomba* no le quite espacio a una *tubería*, por ejemplo, ya que se tratan de elementos que aportan desafíos diferentes en el nivel, y uno no debería interferir en la aparición del otro.

Pasamos, pues, a partir de la siguiente sección, a desarrollar los elementos troncales de nuestros niveles, acompañados de imágenes directamente extraídas de resultados de ejecución.

4.2.2. Descripción de los elementos del nivel

Terreno

El terreno fue lo primero en afrontar en el desarrollo, ya que sirve como base del nivel y sobre él iría el resto de componentes.

Al principio se pensó en un tipo de terreno para todos los niveles, al que llamaremos a partir de ahora terreno montañoso. El terreno montañoso (Figura 4.7) cuenta con desniveles positivos y negativos, y ofrecía un gran atractivo por su verticalidad. Sin embargo, pone varios problemas a la hora de colocar patrones sobre él, y es que si se empezaba a pintar un patrón y el terreno cambiaba de altura mientras se estaba pintando, el patrón quedaba colocado de manera errónea y perdía su sentido (Figura 4.6).



Figura 4.6: Patrón escalera colocado de manera errónea debido a los altibajos del terreno

Se probó entonces en poner una restricción en la que los patrones sólo se pintaran en las zonas llanas de las protuberancias, pero entonces surgía el problema de que los patrones dependían de la forma del terreno para su aparición. Los patrones ya no se colocaban uniformemente sobre el nivel, sino que ahora había que esperar a que encontraran una zona llana suficientemente ancha para poder colocarse, dando lugar a zonas desiertas y otras en las que se concentraban de golpe.

Es por esa razón que se optó por crear un nuevo tipo de terreno, llamado “terreno plano”, que no tendría alteraciones en el terreno y sobre el que se podrían pintar los patrones sin ninguna restricción. El terreno montañoso no se eliminó, sino que quedó relegado a un nuevo tipo de nivel, en el que sólo aparecerían patrones de hoyos (ya que estos modifican el terreno y no dependen de su forma).

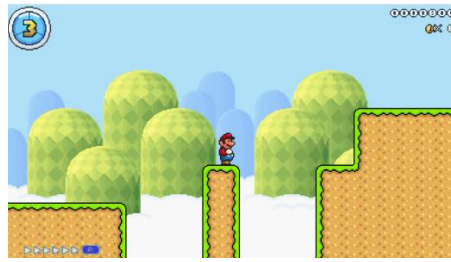


Figura 4.7: Terreno montañoso con patrones de hoyos

Algo que también cambió durante el desarrollo es la forma en la que se decide la anchura total del nivel. Al principio se pensó en una anchura fija e invariable, decidida al principio de la ejecución y que variaría dependiendo de la dificultad elegida para el nivel. Sin embargo, esto traía consigo el problema de que había muchas ocasiones en las que se llegaba al final del nivel sin que se hubieran colocado todos los patrones, quedando los últimos cortados. Por el contrario, podía ocurrir que se colocaran todos los patrones pero quedara un gran terreno vacío en la recta final del nivel, carente de patrones y de interés alguno. Es por esto que se optó por otra manera, que es la que se ha mantenido hasta el final. En esta, el terreno se genera al mismo tiempo que los patrones. De esta forma se crea una regla para todos los niveles: **el ancho de un nivel viene determinado por los patrones que contiene.**

Dado que “Mario Editor” está diseñado para construir niveles de manera visual usando la interfaz del editor, desde el código no hay nociones de lo que es suelo y lo que no. Para solucionar este problema se creó una estructura de datos que almacenara el nivel del suelo para cada posición del nivel. Así, para colocar de manera eficaz los patrones que se dibujarán después, sólo tenemos que referirnos a la estructura, comprobar cuál es nivel del terreno en cierta posición, y dibujar el patrón encima del mismo con la altura que elijamos.

Valles, escaleras y hoyos

Denominaremos este tipo de patrones como “patrones de terreno” para una mejor comprensión de aquí en adelante.

Como buen juego de plataformas, el salto es la mecánica principal de Super Mario. Es por ello que la finalidad subyacente de estos patrones es explotar esta mecánica y poner a prueba las capacidades de salto del jugador en diferentes formas.

Una de estas formas es la de seguir el ritmo del juego. Cuando un jugador está “en el ritmo” de un juego (Smith, Cha, & Whitehead, 2008), efectuar saltos de manera efectiva no requiere sólo un buen cálculo de la distancia de salto, sino también saber elegir el momento justo. Colocar obstáculos de una manera rítmica hace que los saltos sean más sencillos si el jugador sigue el ritmo que marcan. Disponer elementos de una manera rítmica y variada también tiene una función económica, pues proporciona niveles de gran tamaño usando solo unos pocos elementos. En nuestro caso, repitiendo y reorganizando elementos básicos como tuberías, bloques y plataformas, podemos generar niveles largos sin que pierdan el interés.

Cada variante de patrón de terreno tiene asociado un peso, representa la dificultad de pasar a través de él y/o de las amenazas que presenten al jugador. Cada nivel cuenta con una mochila general, en la que se almacenan por igual los tres tipos de patrones de terreno. Cuando comienza la selección, se elige aleatoriamente el tipo troncal del patrón de terreno (valle, escalera u hoyo) y a continuación se elige el sub-tipo semi-aleatoriamente dependiendo de la dificultad elegida para nivel, como hemos explicado en la sección 4.2. Por último, se eligen pseudo-aleatoriamente valores de parámetros inherentes al patrón, como la anchura de los hoyos o la altura de las tuberías. Estos valores se han testado en el proceso de desarrollo de los patrones (explicado en la sección 5.2.3) para evitar obstáculos imposibles de superar.

Los pesos de todos los patrones de terreno están unificados, es decir, si el peso mayor de un patrón de tipo hoyo es X, también lo será el peso mayor de un patrón de tipo valle. Así se evita que un patrón de un tipo pueda ocupar más que el de otro y produzca un desbalanceado en el sistema compartido.

Una vez se ha llenado la mochila, se ordena la lista de patrones por peso ascendente y se empiezan a dibujar en el nivel uno a uno siguiendo dicho orden, eligiendo una distancia mínima de separación entre patrones adecuada. De esta forma se evitan aglomeraciones y se da pie a introducir más tarde otros patrones como enemigos o bloques. Siguiendo este método repetimos sucesivamente hasta recorrer toda la lista.

Enemigos

La aparición de enemigos en el nivel está también regulada por el problema de la mochila, y su peso viene dado por la combinación entre número de enemigos del patrón (tamaño de la horda) y del nivel de enemigo.

Los enemigos se colocan en el nivel después de los patrones de terreno y, muy en el estilo de estos, para la disposición de enemigos en el nivel también se eligen patrones pseudo-aleatoriamente hasta llenar una mochila única para los enemigos. El nivel de los enemigos no es un parámetro dado por el motor, sino una clasificación personal para regular la dificultad en la que separamos los tipos de enemigos en clases por el peligro que presenta su especie, y le asignamos un peso adecuado. Así, para cada tipo de nivel hay 3 niveles de enemigos diferentes. Un ejemplo de nivel 1 sería un Goomba (débil), de nivel 2 un Koopa (normal) y de nivel 3 un Goomba Volador (fuerte).

Algo notable a comentar son los casos en los que se combinan enemigos con patrones de terreno. Dado que la única restricción impuesta a los enemigos es que no se dibujen encima de hoyos, puede darse el caso de que se dibujen dentro del área que comprende un patrón de terreno. Aunque de similar apariencia, este patrón combinado es resultado de la pseudo-aleatoriedad, y no debe confundirse con un patrón que por definición presenta enemigos.

Bloques y *Power-Ups*

Los bloques son uno de los elementos más característicos de la saga Super Mario. En nuestro proyecto, los bloques se subdividen en dos tipos que llamaremos “ladrillos rompibles” y “bloques de contenido”.

Los ladrillos rompibles, como su nombre indica, se rompen cuando Mario los golpea saltando desde abajo. Sirven como plataformas, que añaden algo de verticalidad al nivel y pueden ayudar a llegar a zonas en las que con el salto no es suficiente.

Los bloques de contenido, por el contrario, sirven para proporcionar al jugador alguna ventaja cuando se golpean (la ventaja no es visible hasta que se golpea). Este último tipo tiene un impacto muy fuerte en la regulación de equilibrio del nivel, ya que la diferencia entre aportar ventajas continuamente o aportar muy pocas y separadas entre sí, cambia completamente la

difficultad del mismo. La forma en la que se ha equilibrado esto es regulando, mediante porcentajes, el valor de la ventaja que contiene un bloque conforme se avanza en el nivel. En otras palabras: al principio de un nivel, la probabilidad de que un bloque de contenido albergue una moneda (las cuales no aportan ventajas significativas al jugador) será muy alta, mientras que cerca del final la probabilidad de que contenga un power-up crecerá exponencialmente. Además, el número de bloques de contenido disminuirá según el tipo de mapa que elijamos “FÁCIL”, “NORMAL” o “DIFÍCIL”, respectivamente.

Al margen de estos dos tipos, se han implementado algunos patrones usando bloques, que aportan un reto extra. Un ejemplo de ellos son los bloques musicales, que hacen rebotar a Mario cuando salta sobre ellos. El jugador puede avanzar sin interactuar con ellos, pero si logra encadenar los tres saltos será recompensado con un power-up, oculto a simple vista (Figura 4.9).



Figura 4.9: (a) Tres bloques musicales juntos...



Figura 4.8: (b) ... revelan un power-up oculto

Guía

El patrón Guía se ha implementado de dos maneras principales: mediante el uso de la forma y mediante el uso de coleccionables.

La forma hace referencia al uso de la estructura del mapa para dirigir la mirada del jugador hacia ciertas zonas. Ejemplo de ello es muestra la Figura 4.10, donde los bloques de plataformas apilados de manera vertical apuntan hacia la badera de checkpoint.



Figura 4.10: Guía usando forma

La otra manera en la que se ha implementado la Guía es mediante el uso de monedas. Las monedas en Super Mario son un tipo de coleccionable que proporcionan una vida extra al jugador al recoger cien de ellas. Estas se han programado de manera que se coloquen en formas que predican el movimiento esperado del jugador en cierta zona (Figura 4.11). También se han colocado encima de plataformas para incentivar al jugador para pasar por ellas y quizás descubrir nuevas zonas o vías.

En la Figura 4.12 vemos otro uso efectivo de las monedas. En ella se puede ver cómo el jugador se encuentra en lo que se conoce como un “salto de fe”, es decir, un salto en el que de partida no se ve el suelo. Aquí, el jugador no es capaz de ver si es un hoyo lo que se encuentra bajo sus pies, llevando a una muerte muy injusta en el caso de que si lo hubiera. Es por ello que la función de las monedas alineadas verticalmente es la de incentivar al jugador a bajar siguiendo la hilera, prometiéndole, de manera no verbal, que una zona segura le espera al aterrizar.

Esta misma premisa es usada en la Figura 4.13, donde, en lo que inicialmente parece un hoyo están dispuestas una serie de monedas, de manera inusual. Si el jugador decide aventurarse y seguirlas, será conducido a una tubería oculta a simple vista que le llevará a una zona secreta.



Figura 4.12: Monedas en hilera en un “salto de fe”

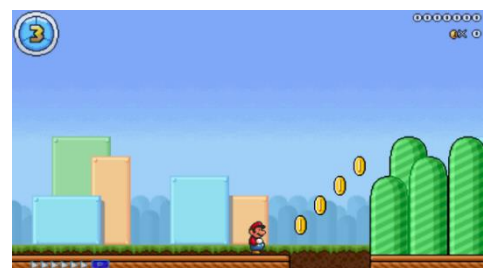


Figura 4.11: Monedas colocadas previendo la dirección del salto de Mario

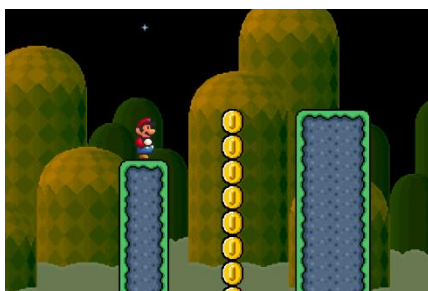


Figura 4.13: (a) Hileras de monedas hacia un hoyo...



Figura 4.13: (b)...conducen a una tubería secreta

Branching

Las tuberías son otro de los elementos característicos y recurrentes en los juegos de Super Mario Bros. A parte de como elemento decorativos, estas suelen usarse para conectar dos zonas del nivel completamente separadas entre sí.

En nuestro proyecto, además de para los valles y hoyos, la manera en la que hemos integrado las tuberías es para crear rutas alternativas en el nivel. De esta manera, se ha decidido que en cada nivel exista una cierta probabilidad de que alguna de las tuberías de los patrones de hoyos o valles sea una tubería especial que lleve a una nueva zona.

Estas tuberías especiales se diferencian de las tuberías normales (aquellas que no conectan con ninguna otra zona) por su color dorado (Figura 4.14), y en ningún momento se le comunica verbalmente al jugador que puede llegar a otra zona a través de estas. La razón de esto es incentivar la detección, por parte del jugador, de patrones inusuales que se salgan de lo visto anteriormente. De esta manera se premia la exploración y la interacción con el entorno (Figura 4.15).

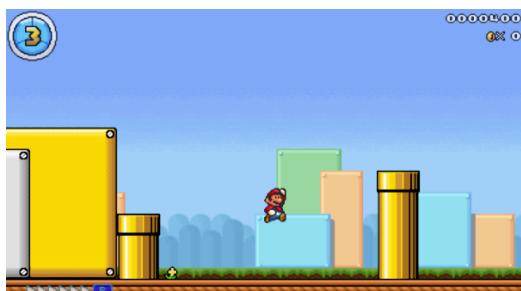


Figura 4.14: Tuberías doradas



Figura 4.15: Monedas y checkpoint recompensan la exploración

La implementación del *Branching* no se limita a conectar con zonas separadas del nivel principal. Para favorecer la continuidad, también se han introducido diferentes caminos a seguir en el nivel principal, en forma de plataformas. Estas nuevas vías, cuando se juntan con otros patrones, pueden dar lugar a nuevas formas de resolverlos. En el caso de la Figura 4.16, vemos cómo el jugador puede atravesar el valle de dos formas. O bien saltando desde el suelo del valle, cuadrando el tiempo del salto para no chocarse con la piraña de la derecha. O bien retrocediendo a la primera tubería, mas baja y accesible, para poder alcanzar las plataformas en forma de seta y salir del valle.



Figura 4.16: Varias alternativas para salir del valle

Riesgo y recompensa

Siendo los *power-ups* o las vidas extra la manera que tienen los juegos de Super Mario de recompensar al jugador, la forma en la que se ha implementado el patrón de riesgo y recompensa es colocando dichos elementos en zonas que presenten cierta amenaza, y sobre las que no sea obligatorio pasar para completar el nivel, quedando pues en un plano opcional.

En la Figura 4.17 podemos ver cómo el jugador puede pasar fácilmente por encima del patrón de escalera para continuar. Sin embargo, si decide bajar y enfrentarse al enemigo será recompensado con un *power-up*.



Figura 4.17: (a) Si el jugador decide bajar al valle y enfrentarse al enemigo...



Figura 4.17: (b) ...se le recompensará con un *power-up*

Sistema de dificultad

Para confeccionar una experiencia retante y con una curva de aprendizaje ascendente, se ha desarrollado un sistema de dificultad. En esencia, su función es la de controlar todos los elementos anteriores y disponerlos en el nivel, mediante porcentajes, de manera acorde a la dificultad elegida. Actualmente el selector de dificultad contiene “FÁCIL”, “NORMAL” y “DIFÍCIL”, y algunos de los factores que controla este sistema son la distancia entre aparición de enemigos, el peso de los patrones elegidos o el número y tipo de *power-up*'s presentes en el nivel.

Elementos decorativos

Por último pero no por ello menos importante, se encuentra el factor decorativo.

Llamamos factor decorativo al conjunto de cambios que se le aplican a un nivel para adecuarlo a una temática o idea.

Estos cambios se comprenden en no-interactuables:

- Fondo y música del nivel
- Estética de los *tiles* que forman el terreno
- Objetos decorativos

E interactuables:

- Tipos de enemigos
- Tipos de power-ups
- Submecánicas

A fecha de este documento, se han implementado X tipos de niveles temáticos diferentes: **Clásico, Desierto, Hielo, Bosque, Volcán, Colinas y Colinas Nocturnas.**

La implementación de los elementos decorativos no-interactuables ayuda a aportar variedad visual a la vez que añade atractivo, vida y personalidad al nivel en sí.

Todos los patrones expuestos en las secciones anteriores se aplican por igual en todos los tipos de nivel. Sin embargo, los cambios interactuables hacen que cada tipo ofrezca enemigos diferentes y submecánicas únicas, y que aporten nuevas formas de resolver ciertas zonas al combinarse con otros patrones.

Un ejemplo de estas submecánicas se pueden ver en la Figura 4.18, donde vemos cómo parte del suelo justo después de un hoyo son arenas movedizas, obligando al jugador a moverse nada más aterrizar para no verse tragado por estas.

Por otra parte en la Figura 4.19, vemos cómo los hoyos se han llenado con lava, adecuándose visualmente al tipo Volcán mientras se mantiene el cometido de los hoyos (derrota si el jugador cae en ellos).

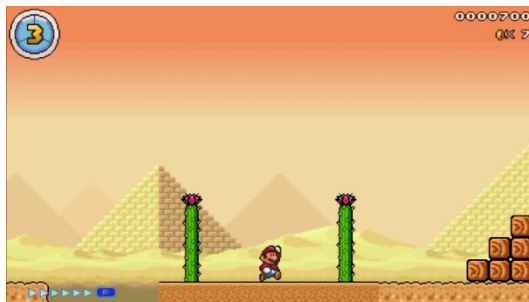


Figura 4.18: Arenas movedizas en mapa tipo Desierto

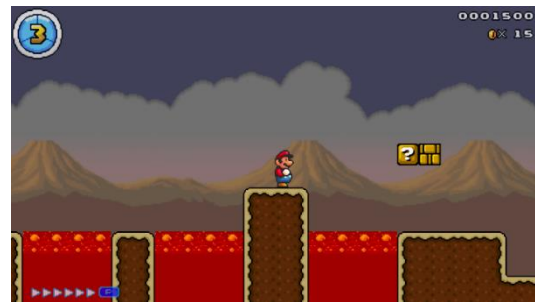


Figura 4.19: Lava en mapa tipo Volcán

4.3. Descripción informática

4.3.1. Proceso de generación

El diagrama de la Figura 4.20 presenta el orden seguido (*pipeline*) para disponer los elementos principales que conforman los niveles.

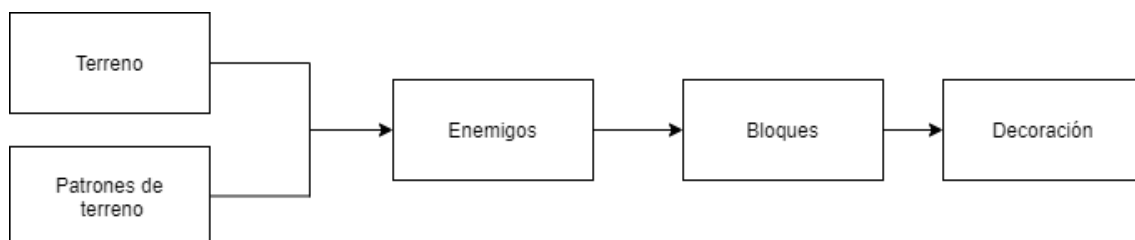


Figura 4.18: Pipeline de generación de niveles

Los niveles se construyen iterativamente, y cada fase representa una pasada al nivel para añadir el elemento que indican. Se han ordenado en este orden según la influencia que ejercen en los

demás, para evitar tener que modificar la estructura de elementos anteriores cuando se entra en una nueva pasada. Siguiendo esta lógica, el terreno y los patrones de terreno son lo primero y lo más importante porque definen la estructura básica del nivel, y la decoración se añade lo último porque son elementos no-interactuables o de bajísima influencia a nivel de estructura en el nivel.

4.3.2. Implementación algoritmo mochila

```
peso = 0
dificultadElegida={FÁCIL,NORMAL,DIFÍCIL}
ALGORITMO MOCHILA

    MIENTRAS peso < dificultadElegida.pesoTotal:

        tipoPatron = ALEATORIO ENTRE {HOYO,VALLE,ESCALERA}

        SEGUN tipoPatron HACER:
            CASO HOYO:
                tipoPatron=HOYO
            CASO VALLE:
                tipoPatron=VALLE
            CASO ESCALERA:
                tipoPatron=ESCALERA
        FIN SEGUN

        subPatron = elegirPatron(tipoPatron.subPatrones,dificultadElegida)

        SI peso + subPatron.peso <= dificultadElegida.pesoTotal
            arrayPatrones.añadir(subPatron)
            peso += subPatron.peso
        SI NO
            SI tipoPatron.subPatrones != VACIO
                tipoPatron.subPatrones.quitar(subPatron)
            SI NO
                SALIR MIENTRAS
        FIN MIENTRAS

        ordenar(arrayPatrones)

        DESDE i = arrayPatrones.inicio HASTA arrayPatrones.fin HACER:
            nivel.colocar(arrayPatrones[i])
        FIN DESDE

FIN ALGORITMO MOCHILA
```

Figura 4.19: Algoritmo de la mochila

El pseudocódigo de la Figura 4.21 muestra la implementación del algoritmo seguido por nuestra mochila de componentes, desarrollada en la sección 4.2.1. En este caso, se trata de la mochila de los patrones de terreno, pero las de los demás componentes siguen la misma premisa de: rellenar mochila hasta alcanzar el peso máximo, ordenar por peso ascendente y colocar en el nivel.

Si el último patrón elegido para completar la mochila se pasa del peso máximo de la misma, este se quita del conjunto de posibles patrones y se vuelve a elegir, manteniendo el sistema de ponderación, hasta encontrar uno que quepa. Si no se puede rellenar con ninguno sin que se sobrepase el máximo, finaliza la búsqueda y se procede a ordenarlos.

ALGORITMO ELEGIR PATRON

```

SEGUN dificultadElegida HACER:
    CASO "FÁCIL":
        probabilidad = ALEATORIO ENTRE {70%,80%,90%}
        SI probabilidad SE CUMPLE
            subPatron = ALEATORIO ENTRE 0 Y tipoPatron.subPatrones.tamaño/2
        SI NO
            subPatron = ALEATORIO ENTRE tipoPatron.subPatrones.tamaño/2 y tipoPatron.subPatrones.tamaño

    CASO "NORMAL":
        subPatron = ALEATORIO ENTRE 0 Y tipoPatron.subPatrones.tamaño

    CASO "DIFÍCIL":
        probabilidad = ALEATORIO ENTRE {70%,80%,90%}
        SI probabilidad SE CUMPLE
            subPatron = ALEATORIO ENTRE tipoPatron.subPatrones.tamaño/2 y tipoPatron.subPatrones.tamaño
        SI NO
            subPatron = ALEATORIO ENTRE 0 Y tipoPatron.subPatrones.tamaño/2
FIN SEGUN

```

FIN ALGORITMO ELEGIR PATRON

Figura 4.20: Algoritmo de selección de patrón

Según el algoritmo de selección de patrones implementado, dependiendo de la dificultad del nivel, la probabilidad de que ciertos patrones se elijan será mucho más alta que con otros. Este criterio de elección se conoce como aleatorio ponderado.

Para referirnos a los subpatrones de cada tipo tenemos un *array* por cada tipo troncal, en el que almacenamos los índices (1,2,3,4...) de los subpatrones, ordenados por dificultad (peso). De esta manera podemos separarlos más fácilmente según el tipo de nivel. Más en detalle, en el tipo “FÁCIL” aumentará la probabilidad de que se elijan los patrones desde el principio del array correspondiente y hasta la mitad del mismo (peso bajo), mientras que en el “DIFÍCIL” serán de mitad para arriba (peso alto). En el tipo NORMAL, por su parte, no habrá ponderación, siendo de criterio aleatorio puro.

Algo a destacar son las llamadas capas de dificultad. Esto significa que, en cada llamada a la función, la ponderación de los patrones cambia aleatoriamente (específicamente entre 70, 80 y 90 por ciento). El motivo de esta implementación es hacer que, aunque sigan teniendo preferencia los patrones del peso que indica la dificultad, no siempre tengan la misma probabilidad de ser elegidos. De esta manera se fomenta la diversidad de niveles y la aparición de diferentes subgrados de dificultad dentro de un mismo tipo.

4.3.3. Estructura del proyecto

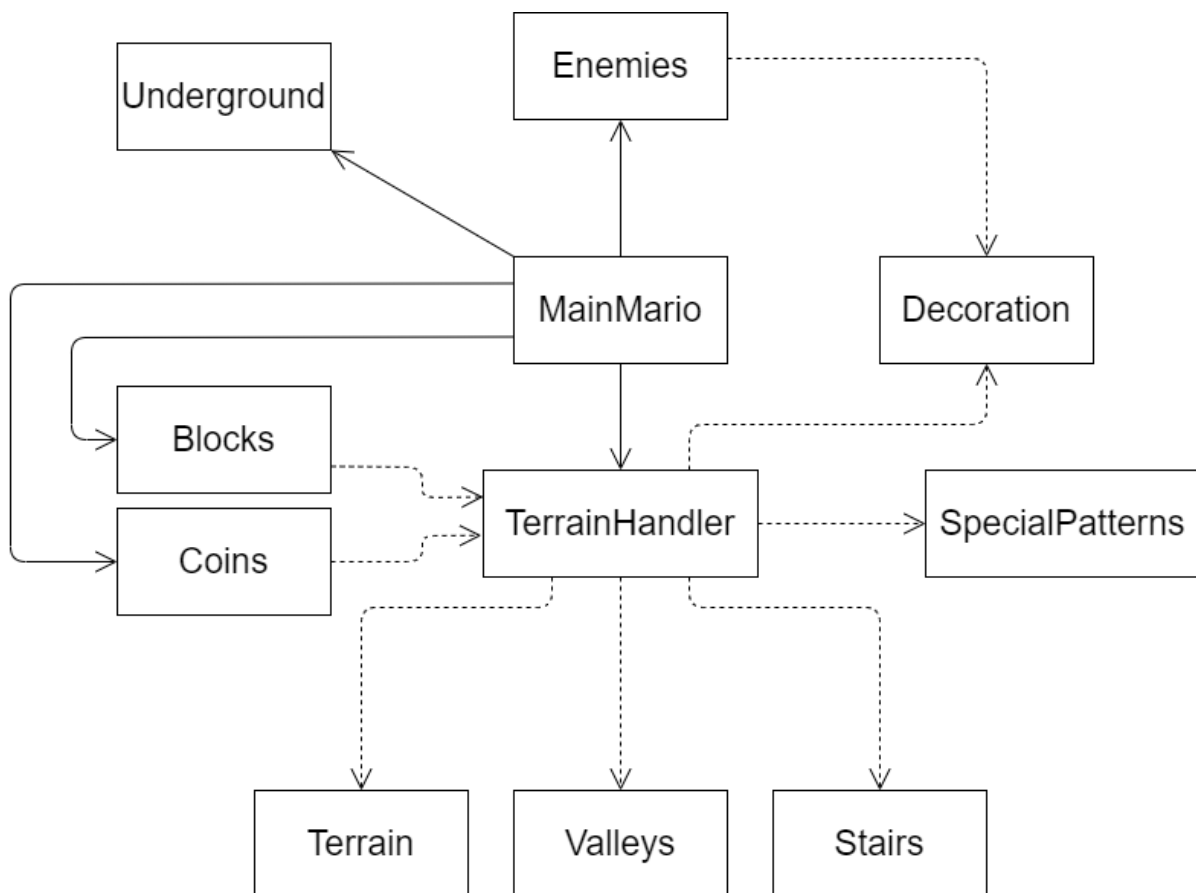


Figura 4.21: Diagrama UML del proyecto

La Figura 4.23 muestra el diagrama UML ⁶ que describe la estructura interna del código del proyecto.

Como se puede apreciar, la clase central y la que genera el archivo .JSON resultante es *MainMario*, y es desde ella que se acceden a todos los componentes que hemos definido para el nivel. *TerrainHandler* se encarga de generar el terreno y de disponer los patrones de terreno

⁶ Lenguaje unificado de modelado (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado

(Valles, Hoyos y Escaleras) sobre él, además de crear las estructuras lógicas que definen el suelo, para poder colocar los Bloques (Blocks) y Monedas (Coins) en las próximas iteraciones.

Dentro de la clase *SpecialPatterns* se detalla la implementación de los patrones abstractos comentados en la sección 3.2.2 (Guía, Forma, Riesgo y Recompensa...).

La clase *Decoration* es usada por *TerrainHandler* y por *Enemies*. En la primera, su función es inicializar los tiles correspondientes dependiendo de la temática del nivel (tiles de arena para tipo Desierto y tiles de nieve para el tipo Hielo, por ejemplo), además de proveer al nivel con assets decorativos, también acordes a la temática, en la última iteración del proceso de generación. En la clase *Enemies*, su función es la de dictar los tipos de enemigos a generar dependiendo, también, de la temática del nivel (las Serpientes de Fuego, por ejemplo, sólo aparecen en el tipo Volcán).

Por último, la clase *Underground* es la encargada de generar la zona secreta a la que se accede a través de las tuberías doradas. Esto se debe a que, al ser una zona separada del nivel principal, es tratada como una nueva escena y necesita una gestión diferente a la del resto del nivel.

En esencia nuestra estructura se podría resumir en un núcleo (o clase) central que genera el archivo jugable (*MainMario*) tras recibir la información obtenida procedualmente de los patrones que conforman el nivel. De escalar este sistema a otros juegos de plataformas, solo habría que sustituir los patrones característicos de Super Mario por los del juego a generar. La implementación de la generación del terreno y la gestión de la decoración, sin embargo, fácilmente podrían ser replicables en otros títulos. El motivo es que son dos elementos de naturaleza universal, presentes en la gran mayoría de juegos del género, lo que hace que se pueda reproducir su estructura entre ellos, sin requerir demasiado esfuerzo en la adaptación.

5. Resultados, análisis y discusión

5.1. Resumen

El propósito de este proyecto se basa en evaluar la utilidad de los patrones de diseño tanto en a) la complejidad de su integración en el proceso de desarrollo y b) la efectividad de los niveles obtenidos. Por esta razón, el análisis de los resultados se basará en el estudio de estos dos puntos por separado

5.2. Evaluación del proceso

En la evaluación del proceso se valorará la complejidad de todas las fases de desarrollo del proyecto, desglosadas a continuación.

5.2.1. Adaptación al motor

La adaptación a “Mario Editor” fue un proceso relativamente sencillo, aunque con algunas cuestiones a solucionar antes de empezar con el desarrollo del generador, pues al no estar pensado para crear niveles de otra forma que no sea usando la interfaz, no es sencillo automatizar el proceso.

La Figura 5.1 muestra la primera captura del proyecto, tomada al inicio del desarrollo. En ella, tiles elegidos del tilemap aleatoriamente se disponen en posiciones aleatorias, sin coherencia alguna. Lo que se pretende mostrar con esto es que en la interfaz del editor no hay nociones de “arriba” o “abajo” o de qué va encima de qué. Esto es así porque inicialmente el motor está maquetado para construir el nivel mediante la interfaz, en la que colocar elementos de una manera lógica se reduce a que las relaciones entre dichos elementos sean visualmente coherentes, como por ejemplo no colocar a un enemigo por debajo de la tierra.

Es por medio de estructuras propias del código, en la que guardamos diferentes niveles de alturas como la del terreno, la de los patrones sobre ellos o la de los bloques y plataformas flotantes, que somos capaces de construir un conjunto de “reglas” que nos ayudan a saber dónde colocar el siguiente elemento



Figura 5.1: Primera captura del proyecto

Otra dificultad presentada por el motor es la forma de identificar los tiles desde el código. Todos los tiles disponibles para usar en los niveles están pre-importados en el motor mediante un tilemap, de manera que la única forma referirnos a ellos desde el código es mediante sus coordenadas del tilemap. Aunque algo tediosa, la única forma de saber qué coordenada corresponde a un tile que queremos usar es colocarlo, mediante la interfaz, en un nivel vacío, guardar el nivel, y ver en el .JSON resultante sus coordenadas.

Aún con esto, se entiende que la aparición de estos pequeños inconvenientes es intrínseca a querer darle un uso diferente al que ofrece, y no se deben al uso de patrones como tal. Era necesario, pues, pasar por cierto periodo adaptativo para ajustarlo a nuestras necesidades.

5.2.2. Generación de terreno

Brevemente comentado en la sección 4.2.2, la generación del terreno pasó por varios métodos hasta llegar al implementado en la versión final. La figura 5.2 muestra una de las primeras versiones del terreno montañoso, en la que la altura del mismo se cambiaba en cada posición de X. Estar saltando constantemente resultaba algo incómodo, por lo que posteriormente se implementó el modelo actual, en la que la altura cambia tras un número aleatorio de X.

La misma figura también sirve para ejemplificar la sección anterior. Antes de que se crearan las estructuras de alturas era muy complicado cuadrar dónde colocar los elementos y se obtenían resultados como el de la imagen.

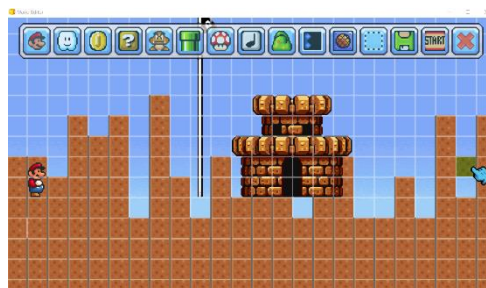


Figura 5.2: Mario está bajo tierra y la bandera está flotando

El terreno base tampoco forma parte de los patrones en sí, y por tanto su implementación y las dificultades que esta pudiera presentar no se deberían tomar en cuenta en la evaluación del modelo de generación por patrones. Pues aunque en la mayoría de juegos del género este presenta una estructura horizontal similar a la de Super Mario, hay juegos que se salen de este paradigma. En *Mega Man* [1], por ejemplo, un mismo nivel puede tener tanto pasillos verticales por los que descender como zonas de avance horizontal.

5.2.3. Generación de patrones

La estructura de los patrones analizados en la sección 4.2.2 se creó a mano. Para ello hubo que programar funciones personalizadas que dispusieran los tiles de una manera fiel a la estructura original de los patrones. Este también fue un proceso relativamente sencillo, y el hecho de que los niveles de Mario Editor estén estructurados en un *grid* facilitó bastante el proceso de automatización, pues permite trabajar con números enteros y hace que podamos colocar estructuras con cómodos bucles⁷.

Por la propia naturaleza de los mismos, hay algunos patrones estructuralmente más complejos que otros, y como era de esperar, las funciones que los crean también lo son. La complejidad de orden lineal⁸, sin embargo, no se ha comprometido en la medida de lo posible. Es importante aclarar, sin embargo, que a pesar de que la eficiencia del código ha sido siempre algo a tener en cuenta en el diseño de funciones, el objetivo del generador no es generar niveles en el menor tiempo posible, por lo que se le ha dado más importancia a la velocidad a la que se generan.

Por último, recalcar que en nuestro proyecto nos hemos enfocado en los juegos de Super Mario, y es posible que en otros juegos la complejidad de programar patrones crezca exponencialmente y el tiempo empleado en generar niveles crezca con ella. Sin embargo, este tiempo siempre va a ser menor que el empleado en construir un nivel a mano, por lo que podemos afirmar que esta técnica es, en términos de tiempo de desarrollo, más eficiente sea cual sea el juego que estemos generando.

⁷ Bucle (programación) (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de [https://es.wikipedia.org/wiki/Bucle_\(programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Bucle_(programaci%C3%B3n))

⁸ Eficiencia algorítmica (s.f.). En Wikipedia. Recuperado el 19 de julio de 2020 de https://es.wikipedia.org/wiki/Eficiencia_algor%C3%ADtmica

5.2.4. Integración de patrones en el terreno

La forma en la que se disponen los patrones en el terreno pasó por varias implementaciones hasta llegar a la actual, en la que se colocan a la vez que el terreno en la primera pasada del *pipeline*. La complejidad aquí reside en que hay que decirle a los patrones cómo se deben disponer en el terreno, pero no vale con colocarlos correctamente sobre el mismo, se deben distribuir de manera que formen un todo con el nivel y no se perciban como elementos individuales e inconexos. Para tratar de conseguir esto:

- Se han ordenado los patrones por peso (dificultad), y
- Se ha elegido un margen adecuado entre patrones para lograr ese “ritmo” que hemos comentado en la sección 4.2.2

5.2.5. Proceso de evaluación

La evaluación se llevó sin demasiada complicación, y se centró en medir los pilares de nuestra hipótesis (funcionalidad y diversidad) en una batería de niveles generados. A pesar de que la evaluación realizada está completa y nos aporta datos significativos, existe mucho potencial para expandir el análisis si se cuenta con más tiempo y recursos.

La evaluación de la funcionalidad se llevó a cabo manualmente y limitada a 100 niveles, pues se considera un mínimo representativo para determinar alguna anomalía en los niveles. De haber contado con una inteligencia artificial, sin embargo, se podrían haber evaluado muchos más niveles tras someterla a un período de entrenamiento para superarlos automáticamente.

La diversidad, al ser algo relativamente objetivo, se ha medido observando las diferencias a nivel estructural que presentan un número determinado de niveles. No obstante, para medir parámetros más subjetivos como el entretenimiento que ofrecen o si aparentan ser niveles generados por una persona, sería necesario contar con una muestra de jugadores externa.

5.3. Evaluación del producto

La efectividad de los niveles obtenidos se determinará según la calidad de su funcionalidad y la densidad de su diversidad.

5.3.1. Evaluación de funcionalidad

Una de las ventajas de usar patrones previamente estructurados y parametrizados nos asegura que la funcionalidad de los niveles generados con ellos no se verá comprometida. En un análisis realizado sobre cien niveles producidos por nuestro generador no se encontró ningún obstáculo imposible, o, en otras palabras, que impidiera al jugador alcanzar el final del nivel.

5.3.2. Evaluación de diversidad

A continuación, se mostrará un análisis llevado a cabo en 3 niveles escogidos al azar, con la intención de medir el factor de diversidad de los mismos. Para ello se han desglosado los niveles en los patrones que lo componen, se han agrupado por pares, y se ha medido el *ratio* de aparición de dichos pares al compararlo con otros niveles.

Para este análisis se han tenido en cuenta los patrones específicos de Super Mario, detallados en la sección 3.3.1, pues son los que mayor impacto tienen en términos de jugabilidad. Usaremos abreviaturas basadas en sus nombres en inglés y en la dificultad del patrón para referirnos a ellos.

De esta manera, los hoyos (*gaps*) serán G, los valles (*valleys*) serán V, las escaleras (*stairs*) serán S, y los enemigos (*enemies*) serán E. Sabiendo esta premisa, para referirnos a un subtipo le añadiremos una cifra entre 1 y 5, representando esta la dificultad inherente al subtipo (en la sección 3.3.1 se encuentran todos los subpatrones de cada tipo troncal, ordenados de menor a mayor dificultad).

Así, por ejemplo:

- Una **horda de tres enemigos** la representamos con **E3**
- Un **hoyo simple** lo representamos con **G1**
- Un **valle con enemigos** lo representamos con **V2**
- Una **escalera valle con hoyo** la representaremos con **V4**

Los niveles escogidos aleatoriamente muestran, según el orden de aparición, este conjunto de patrones cada uno:

Nivel 1:

{S1-E1-G1-G1-G3-G3-S3-E3-S3-G4-S4-G5}

Nivel 2:

{V1-E1-V1-E1-G2-S2-E2-E2-V2-E2-S3-E3-S4-E3-G5-G5}

Nivel 3:

{G1-G1-G2-V2-E2-S2-E3-S2-S2-E3-G2-V4-E3-E3-S4-E4-G5}

Procedemos, pues, a mostrar los resultados de la comparación entre estos tres niveles:

➤ Comparación entre Nivel 1 y Nivel 2

Nivel 1:		
[S1-E1]	[G3-S3]	[S4-G5]
[E1-G1]	[S3-E3]	
[G1-G1]	[E3-S3]	
[G1-G3]	[S3-G4]	
[G3-G3]	[G4-S4]	

11 PATRONES

Nivel 2:		
[V1-E1]	[E2-V2]	[G5-G5]
[E1-V1]	[V2-E2]	
[V1-E2]	[E2-S3]	
[E2-G2]	[S3-E3]	
[G2-S2]	[E3-S4]	
[S2-E2]	[S4-E3]	
[E2-E2]	[E3-G5]	

15 PATRONES

TOTAL → 26 PATRONES

Entre los niveles 1 y 2 se ha encontrado **1 coincidencia** (2 pares de patrones), lo que supone un **7,69%** con respecto tal total de patrones de los dos niveles (26).

➤ Comparación entre Nivel 1 y Nivel 3

Nivel 1:		
[S1-E1]	[G3-S3]	[S4-G5]
[E1-G1]	[S3-E3]	
<u>[G1-G1]</u>	[E3-S3]	
[G1-G3]	[S3-G4]	
[G3-G3]	[G4-S4]	

11 PATRONES

Nivel 3:		
<u>[G1-G1]</u>	[S2-S2]	[S4-E4]
[G1-G2]	[S2-E3]	[E4-G5]
[G2-V2]	[E3-G2]	
[V2-E2]	[G2-V4]	
[E2-S2]	[V4-E3]	
[S2-E3]	[E3-E3]	
[E3-S2]	[E3-S4]	

16 PATRONES

TOTAL → 27 PATRONES

Entre los niveles 1 y 3 se ha encontrado **1 coincidencia** (2 pares de patrones), lo que supone un **7,4%** con respecto tal total de patrones de los dos niveles (27).

➤ Comparación entre Nivel 2 y Nivel 3

Nivel 2:	Nivel 3:
[V1-E1] [E2-V2] [G5-G5]	[G1-G1] [S2-S2] [S4-E4]
[E1-V1] [V2-E2]	[G1-G2] [S2-E3] [E4-G5]
[V1-E2] [E2-S3]	[G2-V2] [E3-G2]
[E2-G2] [S3-E3]	[V2-E2] [G2-V4]
[G2-S2] [E3-S4]	[E2-S2] [V4-E3]
[S2-E2] [S4-E3]	[S2-E3] [E3-E3]
[E2-E2] [E3-G5]	[E3-S2] [E3-S4]
15 PATRONES	16 PATRONES
TOTAL → 31 PATRONES	

Entre los niveles 1 y 3 se ha encontrado **2 coincidencias** (4 pares de patrones), lo que supone un **13,3%** con respecto tal total de patrones de los dos niveles (31).

Como se puede apreciar, la similitud de secuencias de patrones entre niveles es mínima. Y, la secuencias que sí han coincidido, no lo hacen en la misma zona en cada nivel.

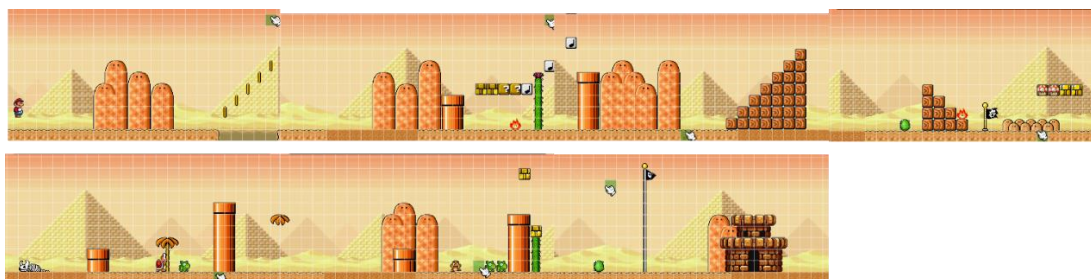
Hay que señalar, además, que ni siquiera estamos teniendo en cuenta los demás componentes de un nivel, como las variaciones del terreno, la disposición de monedas, la posición de bloques y *power-ups*, los *assets* decorativos y el tipo del nivel (desierto, nieve, etc) con lo que esto implica (enemigos diferentes y patrones específicos), entre otros elementos. Todo este conjunto es lo que hace a cada nivel único.

Con el bajo porcentaje de repetición que hemos visto, junto con las casi infinitas posibles combinaciones de secuencias dentro de un nivel, podemos considerar, pues, que nuestro rango de niveles es **diverso**.

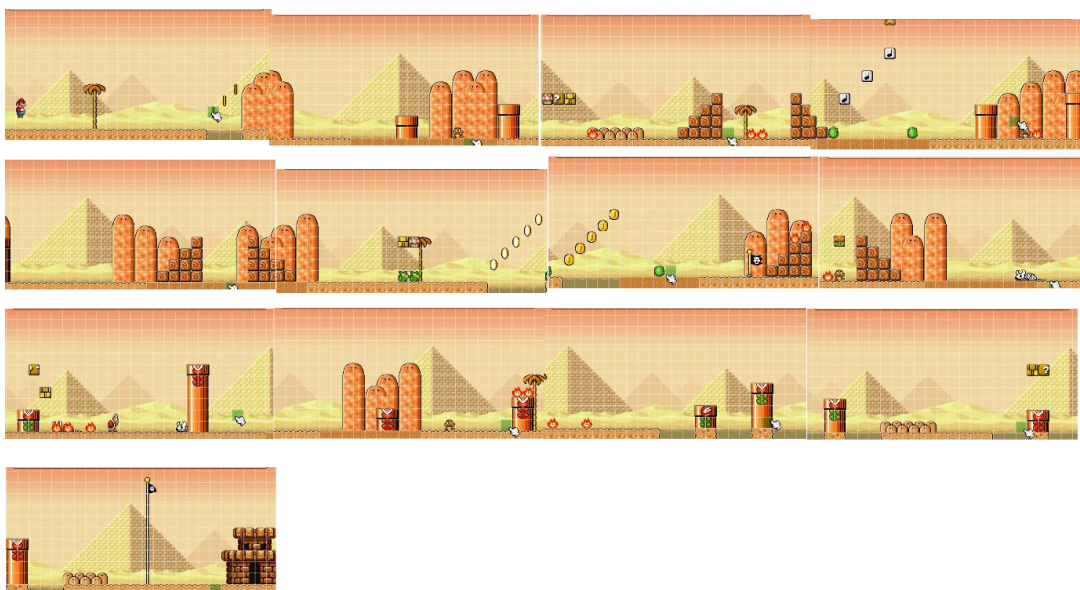
5.3.3. Evaluación de dificultad

Al ser la dificultad una mecánica impuesta por nosotros, se excluye como parte de la evaluación de efectividad de los niveles. Resulta interesante, sin embargo, medir la validez de esta implementación, ya que tiene impacto directo en el *gameplay*. Dada la imposibilidad de visualizar una imagen completa del nivel desde el editor, los siguientes diagramas muestran una serie de capturas de pantalla superpuestas, con la intención de mostrar la estructura de un nivel generado con cada una de las dificultades programadas. Para apreciar mejor las diferencias, las pruebas se han efectuado en el mismo tipo de nivel (Desierto).

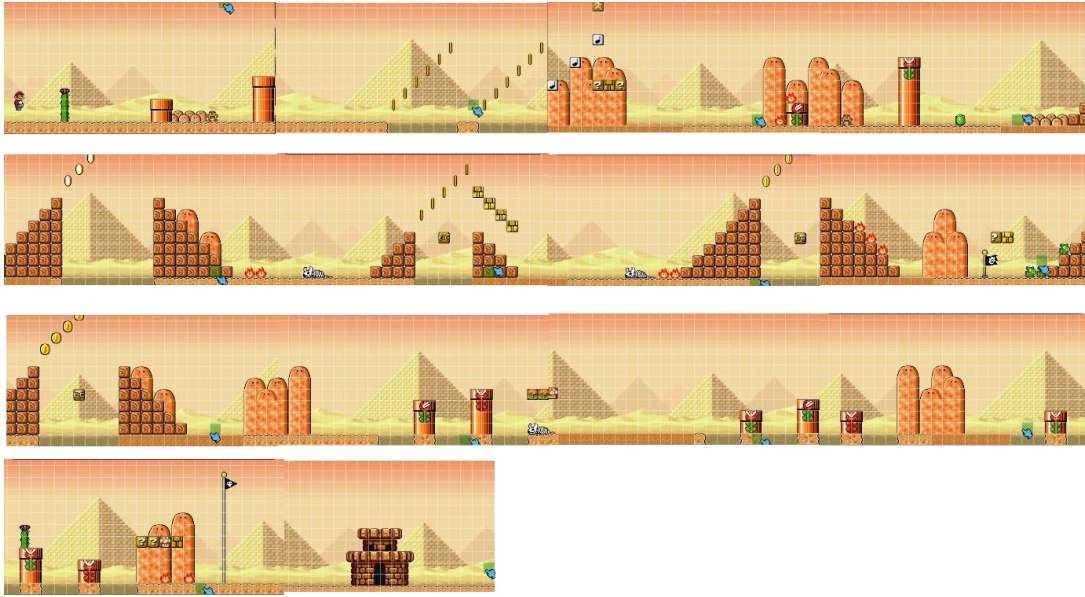
Fácil:



Normal:



Difícil:



Con el aumento de dificultad vemos también un incremento en el peso de los patrones (dificultad que presentan) y en el tamaño total del nivel, así como un decremento en el número y el valor de los bloques de contenido.

6. Conclusiones y trabajos futuros

6.1. Éxito de la propuesta, objetivos e hipótesis

La propuesta, objetivos e hipótesis de este trabajo se formularon con anterioridad en la sección 2. El objetivo de esta sección es visitar cada objetivo y valorar si se han cumplido durante el transcurso del trabajo.

6.1.1. Objetivos

1. Se ha llevado a cabo una búsqueda y análisis de las herramientas y métodos existentes, a fecha del documento, para la generación procedural de niveles de Super Mario. Además se han resaltado los pros y contras de cada una, para terminar exponiendo qué aporta nuestra propuesta a diferencia de ellas. Este análisis se puede encontrar en la sección 3.2.
2. Se han identificado y analizado una batería de patrones tanto de Super Mario como de otros títulos, detallando el por qué de su uso y qué aportan al gameplay. Dicho análisis se encuentra a lo largo de la sección 3.3
3. Se ha creado un generador con éxito, que hace uso de los patrones identificados en el objetivo número 2.
4. El proceso de desarrollo del generador se ha documentado con detalle en la sección 4, señalando los pros y lo contras de las decisiones tomadas, y las dificultades que planteó.
5. La evaluación del generador, así como una valoración de su efectividad están plasmadas en la sección 5.

Todos los objetivos planeados para este proyecto se han completado, pues, con éxito.

6.1.2. Hipótesis

Presentada en la sección 2.3, la hipótesis del proyecto es la siguiente:

Aplicar patrones de diseño a la generación procedural de niveles de Super Mario en 2D proporcionará una serie de niveles diversos y funcionales, y establecerá un modelo que podrá ser escalado con éxito a otros títulos de plataformas 2D para su generación automática.

La funcionalidad de los niveles se evaluó en la sección 5.3.1, y quedó confirmada por el método parametrización de los patrones.

Debido a que en las pruebas se estudiaron un número limitado de casos, no podemos afirmar al 100% que los niveles generados sean considerados “diversos”, pero como hemos expuesto en la sección 5.3.2, se han aportado evidencias que apoyan la diversidad planteada, la cual se puede reforzar en el futuro si se añaden aún más patrones y mecánicas.

Por lo tanto, podemos establecer que aplicar patrones de diseño en este campo de la generación procedural nos proporciona una serie de niveles funcionales, diversos y, en este caso, fieles a la estructura original del juego reproducido (Super Mario). El modelo del que habla la hipótesis no se refiere a una estructura universal que seguir al pie de la letra, sino a una serie de pautas a seguir que se adaptarán a cada implementación en concreto. Con nuestra implementación hemos obtenido resultados satisfactorios. Sin embargo, hasta que no se pruebe en otros juegos y se vea cómo de eficaz resulta la generación de niveles con este método, no podremos afirmar rotundamente la hipótesis. De base, no obstante, tenemos un buen modelo que explotar, con mucho potencial y facilidad para ser escalada (sección 6.4.3).

6.1.3. Propuesta

Por primera vez presentada en la sección 2.1, la propuesta del proyecto dicta así:

Usando a los juegos de Super Mario como modelo, evaluar la utilidad de los patrones de diseño cuando se usan en la generación automática de niveles de plataformas 2D en dos aspectos; tanto en la complejidad que supone su proceso de desarrollo como en la efectividad de los niveles obtenidos.

El proceso de desarrollo está documentado detalladamente a lo largo de la sección 5.2. Tras su evaluación se concluyó que fue un proceso no demasiado complejo, siendo las “mayores” complicaciones del proyecto las decisiones de diseño tomadas para la creación del sistema, y no su implementación en sí.

La efectividad, por otra parte, se detalló en la sección 5.3 y se reafirmó en la 6.1.2 mediante el cumplimiento de la hipótesis.

Ambos aspectos, pues, han sido evaluados cuidadosamente, y han mostrado el beneficio que aportan los patrones de diseño en la generación automática, por lo que podemos concluir que la propuesta se ha cumplido con éxito.

6.2. Limitaciones

6.2.1. Re-skin de patrones

Como hemos visto, la mayoría de los patrones, lejos de tener una apariencia fija, se basan en premisas que admiten varias implementaciones (soluciones). Un ejemplo es la Guía mediante la forma, que si bien la hemos implementado en nuestro proyecto usando bloques de plataformas, se podría lograr también con otros elementos.

Una limitación del proyecto, por tanto, es que al usar los mismos tiles para representar un determinado patrón puede llegar a ser algo repetitivo tras muchas partidas. Hacer diferentes apariencias para cada uno habría tomado demasiado tiempo y se hubiera salido de la idea inicial del proyecto, la cual era evaluar la utilidad de los patrones. Esta limitación es meramente artística y no afecta a la calidad del modelo seguido, pues que funcionen con una apariencia justifica el uso de las demás.

6.2.2. Implementación específica

Por mucho que hayamos conseguido elaborar un modelo flexible y fácilmente escalable a otros juegos, el hecho de que el código de este proyecto se haya enfocado en la construcción de un nivel de Super Mario, y usando el motor *Mario Editor* como herramienta de trabajo, hace que no se pueda reutilizar el mismo para maquetar otros títulos. Aunque habría que hacer grandes cambios en el código para poder lograr esto, es una de las implementaciones que se desean desarrollar en el futuro (sección 6.4.3).

6.3. Contribuciones

6.3.1. Generación aleatoria en juegos de plataformas 2D

Este proyecto pretende arrojar visibilidad a este género y demostrar que se pueden alcanzar buenos resultados adoptando el sistema de diseño por patrones. También recalca factores como

la repetición, ritmo, coherencia, cohesión y continuidad, y su importancia la hora integrarlos en el sistema de patrones para poder seguir generando nuevos niveles de calidad.

6.3.2. Demostración de uso de motor de código abierto para llevar a cabo trabajos de investigación

El hecho de haber realizado este proyecto usando Mario Editor como base pretende mostrar el potencial de los motores de código libre y desestigmatizar su uso como herramientas para la creación de juegos y trabajos de investigación. En nuestro caso en particular, cabe destacar las estructuras JSON, que fueron perfectas como lienzo para trabajar. También el hecho de que estuvieran pre-importadas todas las mecánicas y funcionalidades de elementos como enemigos y bloques fue de gran ayuda, pues nos permitió centrar nuestro trabajo en el uso de estas y no en su implementación. Con todo, a pesar de que hubo que pasar por un pequeño proceso de adaptación para lograr nuestros objetivos, pudimos lograrlos con ayuda del motor.

6.4. En el futuro

6.4.1. Expandir el contenido del generador

El hecho de no poder implementar más tipos de niveles se debe únicamente a las restricciones de tiempo inherentes a la fecha entrega del proyecto. Una de las ideas para llevar a cabo en el futuro es explorar e implementar los tipos de niveles *Subacuático*, *Barco Volador* y *Castillo*, y sus mecánicas asociadas. Estos tres tipos son niveles recurrentes en la saga Super Mario. Sin embargo, por la naturaleza especial de su diseño de niveles (Figura 6.1) rompen con el sistema de patrones que hemos expuesto y por ello no han podido ser implementados en el ámbito de este proyecto.



Figura 6.1: (a) Nivel Subacuático



Figura 6.1: (b) Nivel Barco Volador



Figura 6.2: (c) Nivel Castillo

6.4.2. Generación de niveles on-line

Se ha modelado la dificultad de un nivel de manera externa y bajo las opciones “FÁCIL”, “NORMAL” o “DIFÍCIL”. Encasillar, sin embargo, la habilidad del jugador en estos tres rangos resulta algo limitante.

Para la generación de un nivel en particular, el código del proyecto se ejecuta una sola vez, y antes de que el nivel pueda ser jugado, se almacena la estructura del mismo en un archivo .JSON que leerá “*Mario Editor*”. Mediante esta implementación, no se puede adaptar el nivel mientras se está jugando y técnicas como el escalado de dificultad⁹ quedan fuera de alcance.

Una idea con mucho potencial sería poder recopilar datos que nos dijeran cómo de bien lo está haciendo el jugador en el nivel mientras lo está jugando. Datos como número de muertes, qué enemigo le mata más a menudo, que patrón de terreno se le resiste más o incluso cuánto tiempo tarda en pasarse un nivel, nos permitirían crear una experiencia personalizada, enfatizando aquellos elementos en los que el jugador tiene más problemas para basar el aumento de dificultad en valores objetivos y no en qué patrones tienen más peso.

Por el momento se desconoce si “*Mario Editor*” podría gestionar la modificación de archivos en tiempo de ejecución, por lo que sería necesaria una alteración del motor de no ser el caso. Aún con todo, es un trabajo que se pretende explorar en un futuro.

6.4.3. Escalar el modelo a otros juegos de plataformas 2D

Mediante el cumplimiento de la hipótesis, hemos podido establecer un modelo teórico de disposición de patrones basado en el problema de la mochila, cuya flexibilidad permitiría su escalado a otros juegos.

Lo ideal sería poder crear un motor propio mediante el cual poder crear o importar patrones y tras parametrizar valores como el peso de dichos patrones, la capacidad de la mochila o la forma del terreno, ser capaces de generar niveles de dicho juego siguiendo este modelo.

Ejemplos de juegos que se beneficiarían mucho de este sistema son la saga *Donkey Kong* y *Mega Man*, debido a sus marcados patrones y a la importancia de los mismos en el gameplay (Brown, 2018)

⁹ García, J. (21 de abril de 2015) *La curva de dificultad en videojuegos*. Obtenido de <https://es.ign.com/reportaje/92847/feature/la-curva-de-dificultad-en-videojuegos> el 19 de julio de 2020

6.5. Conclusión

En este proyecto hemos analizado el potencial de los patrones de diseño en la generación de contenido procedural, y presentado una forma de crear niveles de Super Mario usando esta premisa.

Aunque todas las implementaciones se han llevado a cabo con éxito, el resultado es algo limitado. No obstante, se pretende que el resultado obtenido sirva para arrojar algo de visibilidad al género plataformero en el mundo de la generación automatizada.

A pesar de que el contenido generado automáticamente siga cojeando en algunas áreas en las que el generado por humanos no, es beneficioso tomar iniciativa e intentar integrar la generación automática en juegos existentes que precisamente exhiban estas áreas, como hemos hecho en este trabajo con el género de plataformas 2D. Sólo de esta manera, podremos seguir expandiendo los horizontes de la generación procedural en los videojuegos y avanzando en la investigación de esta potente herramienta.

7. Referencias

- Alexander, C., Silverstein, M., & Ishikawa, S. (1977).
- Bjork, S., & Holopainen, J. (2005). *Patterns in game design (game development series)*.
- Bork, S. (s.f.). *Gameplay design patterns collection*. Obtenido de <http://virt10.itu.chalmers.se/index.php/Category:Patterns> el 19 de julio de 2020
- Brown, M. (16 de Junio de 2017). *Donkey Kong Country: Tropical Freeze - Mario's Level Design, Evolved | Game Maker's Toolkit*. Obtenido de <https://www.youtube.com/watch?v=JqHcE6B4OP4> el 19 de julio de 2020
- Brown, M. (24 de Octubre de 2018). *How Mega Man 11's Levels Do More With Less | Game Maker's Toolkit*. Obtenido de <https://www.youtube.com/watch?v=nYxHMZX6lN8> el 19 de julio de 2020
- Brown, M. (6 de Abril de 2018). *How to keep players engaged (Without being evil) | Game Maker's Toolkit*. Obtenido de https://www.youtube.com/watch?v=hbzGO_Qonu0 el 19 de julio de 2020
- Clark, N., & Anthropy, A. (2014). *A game design vocabulary: Exploring the foundational principles behind good game design*.
- Dahlskog, S., & Togelius, J. (2012). *Patterns and Procedural Content Generation*.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*.
- Garcia, J. D. (5 de Abril de 2016). *C++ en la industria del videojuego*. Obtenido de <https://www.youtube.com/watch?v=FDr6nywobhw> el 19 de julio de 2020
- Guzdial, M., & Riedl, M. O. (2016). *Toward Game Level Generation from Gameplay Videos*.
- Hello Fangaming. (s.f.). Obtenido de <https://hellofangaming.github.io/> el 19 de julio de 2020
- Portoff, S. (13 de Junio de 2015). *MarI/O - Machine Learning for Video Games*. Obtenido de <https://www.youtube.com/watch?v=qv6UVOQ0F44&t=225s> el 19 de julio de 2020
- Smith, G., Cha, M., & Whitehead, J. (2008). *A framework for Analysis of 2D Platformer Levels*.
- Summerville, A., & Mateas, M. (2016). *Super Mario as a String: Platformer Level Generation Via LSTMs*.
- Thompson, T. (30 de Junio de 2015). *Researching Super Mario Bros. Level Design | AI and Games*. Obtenido de https://www.youtube.com/watch?v=t_zXpKlccRE&t=392s
- Togelius, J., Karakovskiy, S., & Baumgarten, R. (2010). *The 2009 Mario AI Competition*.
- Wagner, R. (1851). *Ópera y drama*.
- Wikipedia. (s.f.). *Problema de la mochila*. Obtenido de https://es.wikipedia.org/wiki/Problema_de_la_mochila el 19 de julio de 2020