Magic Forest: Una implementación de la Wave Function Collapse

Magic Forest: A Wave Function Collapse Implementation

Martín St. John Clarke Bideau
Estudiante de la Licenciatura en Artes Digitales
División de Ingenierías del Campus Irapuato-Salamanca
Universidad de Guanajuato

martinst.johnclarke@gmail.com
m.stjohnclarkebideau@ugto.mx

Resumen

Innumerables videojuegos utilizan la generación de contenido de manera procedimental para su creación, y resulta útil y a veces indispensable su uso en esta área, por los innumerables elementos que se benefician de esta clase de generación. Desde texturas, modelos en 3D, terrenos, mapas, mecánicas, e inclusive estructuras que forman parte fundamental de su valor artístico o como obra de entretenimiento. De allí parte la motivación de esta investigación, ya que su desarrollo pude ayudar a comprender el origen de la belleza en aquello en lo que no interviene completamente el ser humano.

El objetivo del proyecto es la investigación e implementación de un algoritmo de generación procedimental con un enfoque al ámbito de los videojuegos y su valor como obras.

Palabras clave: Generación procedimental; Wave Function Collapse; Celda; Tile; Colapso.

Introducción

El contenido creado por procedimientos (de manera procedimental), se caracteriza por cuya generación se da a partir de algoritmos y no de manera puramente manual. Esto ofrece una alternativa para la creación, que puede resultar imprescindible y precisa en algunos casos, o bien innecesaria y limitante en algunos otros, debido a su naturaleza parcialmente automática. Dicha naturaleza quita control manual a varias características del producto generado, aspecto que no es necesariamente positivo o negativo, simplemente revela que conocer las virtudes y limitaciones de esta técnica, también es clave a la hora de enfrentarse a un proyecto interactivo.

Entonces, la investigación de algoritmos procedimentales enriquecerá tanto un aspecto técnico, como su entendimiento en relación con obras artísticas y de entretenimiento. Con este objetivo, el algoritmo principal elegido para el desarrollo de este proyecto es el llamado Wave Function Collapse (WFC) o colapso de función de onda (Maxim Gumin, 2016), que puede ser usado, entre otras aplicaciones, para la generación de entornos y escenarios en dos y tres dimensiones. A grandes rasgos, funciona interconectando elementos simples, ya sean pixeles, imágenes de partes de figuras u objetos, o partes de modelos en 3D para crear elementos más grandes, complejos y con sentido.

Se eligió usar este algoritmo por la belleza que se percibió de imitar la naturaleza, en el sentido de proponer restricciones precisas a elementos y sus interacciones con otros, pero encargar su generación masiva o diversa a una cierta aleatoriedad.

Existen muchos ejemplos de implementaciones interesantes del algoritmo elegido. A continuación, se muestran imágenes de algunos de ellos.

Figura 1. Ejemplo de implementación en 3D del algoritmo WFC. (Imagen tomada del videojuego Townscaper)

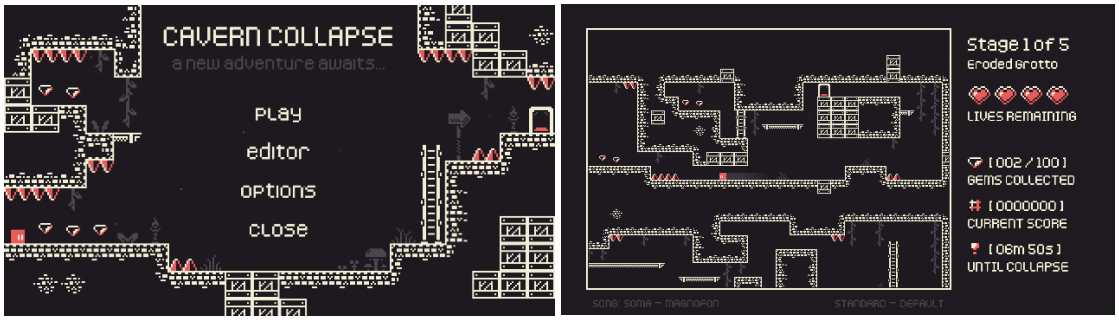

Figura 2. Ejemplo de generación de niveles en 2D con el algoritmo WFC. (Imagen tomada del videojuego Cavern Collapse)

Metodología

Conceptos básicos del algoritmo

El algoritmo Wave Function Collapse puede ser implementado de muy diversas maneras. La manera en la que fue concebido por Maxim Gumin (2016) involucra una imagen de referencia que permite generar imágenes de salida conservando los patrones, característica de la cual se prescindió en la implementación presente, sin embargo, la esencia del funcionamiento de este algoritmo radica en restringir posibilidades entre elementos simples. Por ejemplo, tomando prestadas algunas ideas de la explicación de Robert Heaton (2018) sobre dicha función, planteemos una cuadrícula, y en cada celda de esta cuadrícula, podemos tener cualquiera de los siguientes tres elementos: agua, arena y pasto. En este momento cualquiera de estos tres elementos puede estar en cualquier celda, por lo que cada uno estará escrito en cada celda, solo como posibilidad.

Figura 3. Cuadrícula con cada posibilidad explícita y ninguna celda definida

Teniendo estos elementos, podemos imponer algunas reglas.

1. El pasto solamente puede estar al lado del pasto y de la arena.
2. El agua solamente puede estar al lado del agua y la arena.
3. La arena puede estar al lado de cualquiera de los tres elementos.

De esta manera, si se decide arbitrariamente asignarle agua a alguna de las celdas, las posibilidades que se escribieron en las celdas adyacentes, se limitan.

Figura 4. Cuadrícula con la celda central definida en agua

Vemos que, como el pasto no puede colindar con el agua, el pasto deja de ser una posibilidad para las celdas colindantes y solamente podrías poner, o pasto, o arena a sus lados.

Si con esta misma lógica se siguen decidiendo arbitrariamente entre las opciones que quedan y luego limitando las opciones adyacentes, podría quedar una imagen de la siguiente manera:

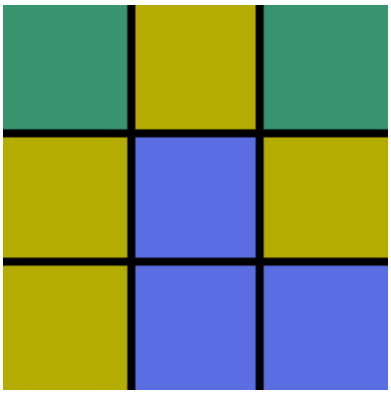

Figura 5. Cuadrícula con cada celda definida

Y como se puede ver, cada celda cumple con las restricciones planteadas. Ningún cuadro de agua tiene pasto al costado.

Así, esencialmente, funciona la Wave Function Collapse. Tienes una serie de opciones, que a partir de ahora se nombrarán “tiles”. Por otro lado, tenemos una cuadrícula con celdas, que, en principio, pueden contener cualquiera de los tiles. Solo en la primera iteración del algoritmo, se selecciona aleatoriamente cualquiera de

las celdas para poder empezar, y se elige aleatoriamente alguno de los tiles que dicha celda posee, lo que se suele llamar “colapsar la celda”, pues pasa de un estado en el que puede ser cualquiera de los tiles restantes, a un estado colapsado, en el que ya se definió de manera aleatoria un único tile que será visualizado en pantalla. Una vez se colapsa una celda aleatoria, se propaga el efecto del colapso, eso quiere decir que, de las celdas adyacentes a la que acaba de colapsar (arriba, abajo, izquierda y derecha), se eliminan las posibilidades que no son compatibles con el tile definido por el colapso, como en el caso anterior que, al colapsar la celda central en agua, las celdas adyacentes tuvieron que eliminar la posibilidad de pasto. Una vez propagado el efecto de la celda colapsada, se busca la celda con la menor cantidad de tiles posibles, y se repite el procedimiento con esa celda. Colapsa una celda, se eliminan los tiles adyacentes que son incompatibles con el tile de la celda colapsada y se busca la celda con menos opciones restantes, así hasta que la cuadrícula haya colapsado por completo.

Las restricciones entre celdas se pueden plantear de diversas maneras. Una de ellas es la antes expuesta, determinada por condiciones, pero hay otra opción cuyas restricciones se dan a partir de los llamados “sockets”. Básicamente se trata de etiquetar cada lado de cada tile, ya sea con un número, un carácter o una cadena de caracteres. El punto es identificarlo, pues únicamente se permitirá conectar un tile con otro si los lados a unir tienen el mismo identificador.

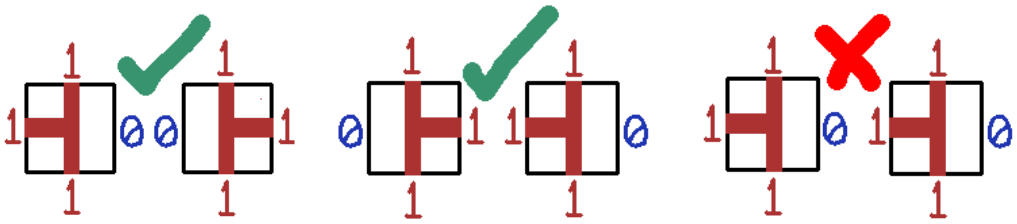

Figura 6. Ejemplo de tiles con dos sockets posibles (0 y 1) y sus conexiones permitidas

Con dicha condición, se eliminarían tiles si los lados que están en contacto no coinciden. Así, ejecutando el procedimiento antes descrito, podría generarse algo de este estilo:

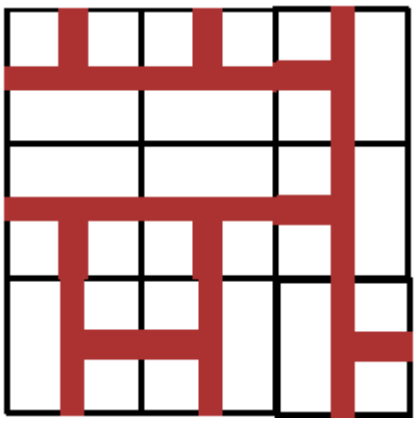

Figura 7. Cuadrícula con cada celda definida

Dicho etiquetado se puede hacer de manera manual o automática, más automática cuando se trata de modelos en 3D pues se puede detectar la posición de los vértices que hay en cada lado de una celda cúbica.

Dada la adaptabilidad y practicidad de este método, se determinó que este sería usado para la implementación del algoritmo.

Implementación

Para su implementación se utilizó la plataforma para desarrollar videojuegos Unity.

En código, se crearon tres clases principales por cada uno de tres los elementos descritos anteriormente que son clave para el funcionamiento de WFC: tile, celda y cuadrícula. Para efectos didácticos llamaremos a dichas clases Tile, Cell y Grid respectivamente.

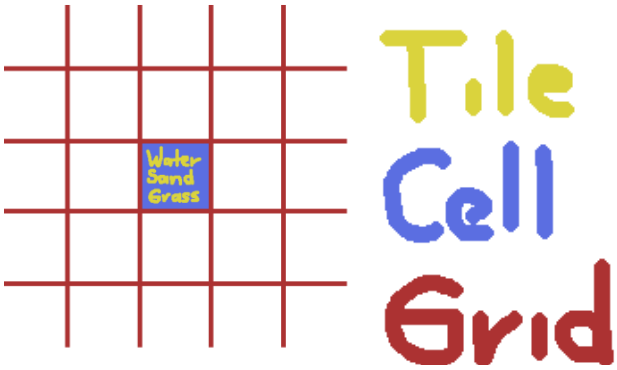

Figura 8. Representación visual del significado de Tile, Cell y Grid

Entonces, una instancia de la clase Tile representaría una de las posibles imágenes que podría tener una celda. Esta clase contiene la imagen que se mostrará y cada uno de los sockets correspondientes.

Una representación visual de una instancia de la clase Tile podría lucir de la siguiente manera

Figura 9. Representación visual de una instancia de la clase Tile

En esta implementación, cada socket sería una cadena de caracteres compuesta por la inicial de el o los elementos que aparecerían en un socket, de manera que, si hay un elemento en uno de los lados de un determinado tile, se le pone un carácter en mayúscula que lo defina. Por ejemplo, en el tile de la figura 9, cuando hay pasto, se etiqueta ese lado como G, de Grass, y cuando hay agua, se etiqueta como W, de Water. Cuando ambos elementos están en un mismo lado, se etiqueta a dicho lado con la inicial de ambos elementos en el orden en el que están, de izquierda a derecha y de arriba a abajo. Por ejemplo, el socket superior de la figura 9 se llamaría “GW”. De manera que esta Tile quedaría etiquetada como: [GW, W, GW, G] en el orden [arriba, derecha, abajo, izquierda].

Cada instancia de la clase Cell contiene una lista con cada instancia existente de la clase Tile, que contiene su imagen correspondiente, sus sockets y su peso. La clase Cell tiene un método para colapsar, eligiendo de manera aleatoria entre los tiles que quedan tomando en cuenta el peso que tengan los tiles. El peso es un simple número que se le asigna a un tile y determina qué tan probable es que se elija dicho tile en relación con los pesos de los otros tiles restantes.

Finalmente, la clase Grid se encarga de gestionar todas las celdas. Al ser Grid la clase más externa, tiene la capacidad de analizar celdas entre sí. Esta clase genera una matriz con las dimensiones que se desea que

tenga tu cuadrícula, según cuantas celdas quieres que tenga de ancho y de alto. Se genera una instancia de la clase Cell por cada espacio en la matriz antes mencionada y se almacena en esta, de manera que se crea una cuadrícula. En pantalla se acomodan bajo este mismo orden, como matriz.

Teniendo una matriz llena de Cells, se comienza el análisis.

1. Hallar la celda con la entropía más baja.

    - Hace falta precisar qué se entiende por entropía en esta implementación. Como se mencionó en la sección “Conceptos básicos del algoritmo”, para decidir qué celda colapsar, se requiere saber qué celda tiene menor cantidad de Tiles restantes, pues de esta manera el algoritmo es menos propenso a llegar a una situación de contradicción, en la que una celda se queda sin tiles posibles gracias a que las celdas adyacentes se las retiraron. En este caso, decir que la celda con la menor cantidad de tiles posibles es la celda con menor entropía es siempre correcto, sin embargo, como se mencionó antes, ahora cada tile tiene también un peso, es decir, una probabilidad determinada de que se elija aleatoriamente. Este peso afecta a la entropía, pues a mayor probabilidad exista de que un tile específico colapse, menos incertidumbre hay, es decir, menos entropía. De manera que ahora se calcula la entropía de una celda en función de la suma de los pesos de sus tiles restantes.

    Para seleccionar la celda con menor entropía, se repasan todas las celdas que aún no han colapsado comparando sus entropías. Este proceso devuelve como resultado la primera celda con la entropía más baja. Para el caso en el cual exista más de una celda con la misma mínima entropía, se repasa una vez más la matriz, obteniendo ahora una lista con las celdas que comparten la misma mínima entropía. Finalmente se elige aleatoriamente una celda de esta lista, obteniendo así, la o una de las celdas con la entropía mínima en ese momento.

2. Una vez obtenida la celda con menor entropía, se colapsa.

3. Al haber colapsado una celda, esta se queda con un solo tile, y ahora hará falta propagar este efecto, pues como ya se vio, reducir los tiles de una celda puede afectar los posibles tiles de las celdas adyacentes, ya que algunas no serán compatibles con el o los tiles que quedan.

    Dicha propagación funciona de la siguiente manera:

    Partimos de una celda colapsada, lo que quiere decir que la celda perdió tiles y por lo tanto hace falta analizar su efecto en las demás.

    1. Se crea una lista que almacenará las celdas afectadas, dato que cobrará sentido en los siguientes puntos.
    2. Se agrega la celda colapsada a la lista como su único elemento inicial.
    3. Se entra a un ciclo que itera cuantas veces como elementos haya en esa lista.
    4. Dentro de ese ciclo, hay un método que recibe una instancia de la clase Cell como parámetro, al cual se le enviará el elemento actual de la lista (dado por las iteraciones del ciclo), en este caso, la celda colapsada.
        - Dicha función compara los tiles de la celda enviada con los tiles de sus celdas adyacentes, eliminando aquellos tiles que ya no son compatibles con la celda mandada.
        - La clave es que, si se elimina un tile de una celda adyacente, esta habría sido afectada. Sus posibilidades se reducen y por ende conviene revisar si los tiles ahora ausentes provocan la eliminación de algún tile en alguna celda adyacente a la afectada, por lo que, si hay una celda afectada, se guardará en la lista mencionada en el punto 1.
    5. De esta manera, en la siguiente iteración del ciclo mencionado en el punto 3, se analizará ahora aquella celda que perdió tiles. Si una nueva celda se ve afectada gracias a este análisis, se agregará al arreglo de celdas afectadas para ser analizada y se repetirá el ciclo hasta que se analice cada celda afectada directa o indirectamente por el colapso de la celda inicial.

    En resumen, un ciclo va a analizar cada celda afectada de una lista, dicho análisis puede provocar que se eliminen tiles de celdas adyacentes y por lo tanto que añada una nueva celda a la lista y se prolongue el ciclo hasta que se propague por completo el efecto.

Por último, siempre existe la posibilidad de que se llegue a una incongruencia dentro de este algoritmo. Es decir, que se llegue a un punto en el que no sea posible colapsar una celda debido a que las celdas

adyacentes a ella restringieron todas y cada una de sus posibilidades. Esto provocaría un hueco en nuestra imagen generada. Para evitar esto, en el momento en el que una celda queda con cero tiles posibles, se restablece por completo dicha celda y tres a su alrededor, formando un cuadrado de 7 por 7 celdas cuyo centro es la celda que llegó a cero. Este restablecimiento de las celdas involucra que las celdas mencionadas dejarán de estar colapsadas si lo estaban y se les devolverá cada uno de los posibles tiles que se tenían en un inicio. Para evitar que esto provoque nuevos errores, se propaga el efecto de todas las celdas colapsadas de la cuadrícula, lo que resta los tiles incompatibles a las celdas recién restablecidas. Finalmente vuelven a colapsar según su entropía, de manera que se soluciona el riesgo a obtener una imagen incompleta.

Resultados

Se logró una implementación del algoritmo Wave Function Collapse en Unity con un ejecutable y un funcionamiento correcto y preciso. En la versión final no se ha detectado ningún comportamiento inesperado ni erróneo. Los errores que llegan a ocurrir al haber contradicciones son naturales y se resuelven automáticamente sin complicaciones gracias a las funciones de seguridad que fueron aplicadas.

El apartado visual de la implementación del algoritmo cuenta con una serie de 28 tiles creados especialmente para este proyecto, que se interconectan para formar una ambientación semejante a la de un bosque o un pantano con cascadas.

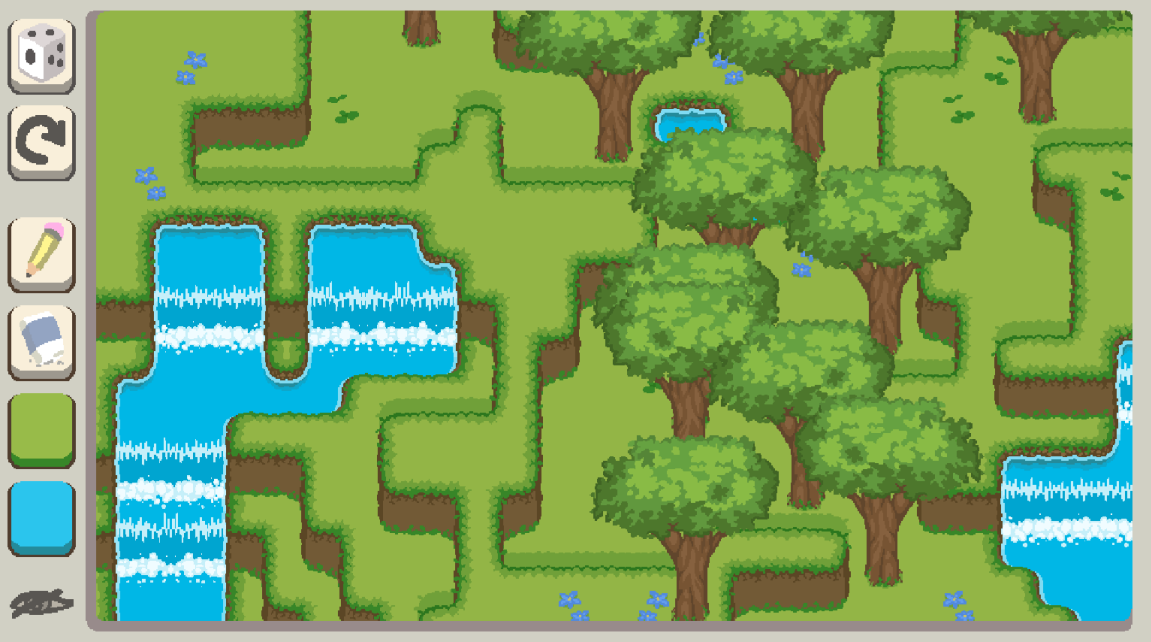

Figura 10. Captura del programa desarrollado para el algoritmo, siendo ejecutado

Además de la implementación del algoritmo, se creó una interfaz de usuario que permite experimentar e interactuar con su funcionamiento de las siguientes maneras (Vea la figura 10):

1. El botón con ícono de dado te permite generar o terminar de generar el entorno de manera aleatoria al ser presionado.
2. El botón con ícono de flecha circular restablece todo el entorno. Destruye la cuadrícula en el estado en el que esté y genera una nueva y vacía, pero no vuelve a generar un entorno hasta que se lo indiques con el dado.
3. El botón con ícono de lápiz permite al usuario colapsar celdas no colapsadas a voluntad al dar clic en la deseada.
4. El botón con ícono de borrador permite descolapsar y restablecer los tiles de cualquier celda al dar clic sobre ella.

5. Tanto el botón verde como el azul te permiten definir donde se generará un elemento elegido. El verde define elementos con pasto y el azul elementos con agua.
    - En caso de que traces algo que pueda generar una contradicción, es posible que se deshaga lo que trazaste y no se genere el elemento que definiste.
    - Tienen un sistema de seguridad para evitar contradicciones que, al dibujar el elemento deseado, restaura y descolapsa todas las celdas cercanas que contengan tiles no deseados para la correcta generación del elemento.
    - Una vez que dibujas con los elementos deseados, hace falta presionar el botón del dado para que se genere la imagen.
6. El ícono con forma de ojo permite ver el número de tiles restantes en cada una de las celdas.

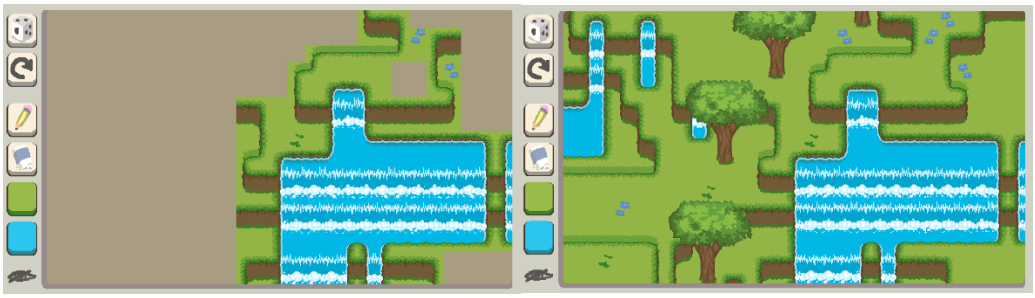

Figura 11. Captura del proceso de generación de un escenario y su resultado final

Figura 12. Captura del sistema de dibujo de agua y pasto y su resultado al presionar el dado

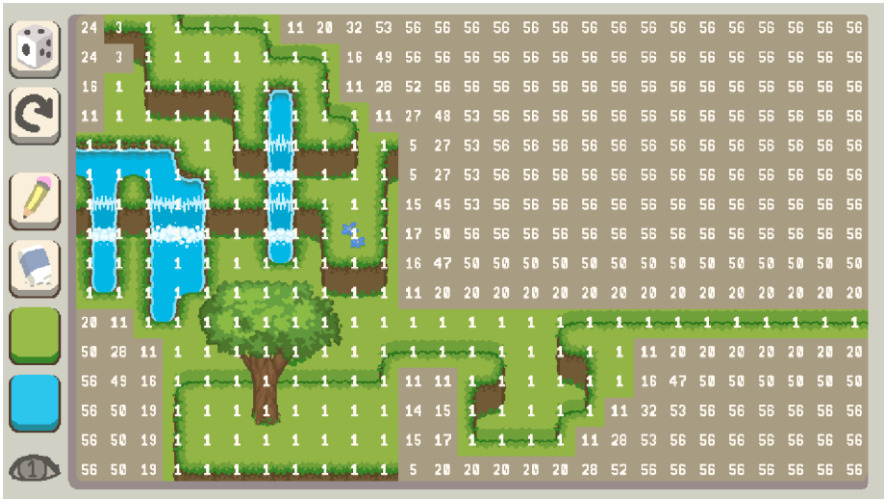

Figura 12. Captura del programa con los números de tiles restantes en cada celda siendo visualizados

Conclusión

Si bien la implementación para este proyecto se trata de un programa interactivo que pudiera no ser considerado como un videojuego (o por lo menos no uno muy complejo), puede resultar valioso para una comprensión de la naturaleza de la generación de contenido procedimental.

Se trata de un algoritmo cuyas bases se asientan en un concepto elegante e intuitivo, que se puede relacionar con videojuegos o juegos tales como el sudoku o el buscaminas, cuya esencia radica también en restringir las posibilidades para hallar un resultado que, en su totalidad satisfaga las reglas básicas que se impusieron en su origen. En un primer momento, uno no sería capaz de relacionar un juego en el cual ningún número se puede repetir en línea vertical u horizontal, con la generación de un entorno de dos o tres dimensiones con sentido y estética, y es ahí donde resulta más fácil apreciar la belleza de aquello que no se diseña explícitamente. Un planteamiento de reglas lógicas y creativas que hacen que una cuadrícula de números se ordene de una forma específica parece ahora ser la clave para ordenar piezas que pueden generar infinitas configuraciones que tratan de imitar a la naturaleza. Entender ahora al sudoku o al buscaminas con un filtro de paisajes, patrones o figuras puede resultar revelador, y la implementación de una interfaz amigable en este algoritmo, permite relacionar estos conceptos de una manera intuitiva y explorar su funcionamiento, además de plantear preguntas que encaminan a entender la esencia de lo divertido, lo aburrido o lo emocionante, pues si un algoritmo es capaz de interconectar piezas diseñadas y generar algo visualmente bello, ¿qué se necesita para lograr eso mismo con aquello que divierte o entusiasma? Queda conocer los límites de este tipo de generación y saber cómo aprovechar sus virtudes.

Bibliografía/Referencias

Gumin, M. G. (s. f.). GitHub - mxGMn/WaveFunctionCollapse: Bitmap & Tilemap generation from a single example with the help of ideas from quantum mechanics. GitHub. https://github.com/mxgmn/WaveFunctionCollapse

The wavefunction collapse algorithm explained very clearly | Robert Heaton. (2018, 17 diciembre). Robert Heaton. https://robertheaton.com/2018/12/17/wavefunction-collapse-algorithm/

Martin Donald. (2020, 31 julio). Superpositions, sudoku, the wave function collapse algorithm. [Vídeo]. YouTube. https://www.youtube.com/watch?v=2SuvO4Gi7uY

Cheng, D., Han, H., & Fei, G. (2020). Automatic generation of game levels based on controllable wave Function collapse algorithm. En Lecture Notes in Computer Science (pp. 37-50). https://doi.org/10.1007/978-3-030-65736-9_3