VOLUMEN 21
XXVIII Verano De la Ciencia
ISSN 2395-9797
www.jóvenesenlaciencia.ugto.mx

Generación de contenido por procedimientos en entornos artísticos y computacionales

Procedural content generation in artistic and computational environments

Jesús Bertani Ramírez*[1], Juan Aguilera Huerta**[2], Alejandro Martínez Ledesma ***[3], Martin St. John Clarke Bideau***[4], Luis Eduardo Domínguez Hurtado***[5], Walberto Michel Durán Sierra***[6], Uriel Haile Hernández Belmonte ****[7], Rocío Alfonsina Lizárraga Morales****[8]

* Lic. en Computación Matemática, División De Ciencias Naturales y Exactas, Campus Guanajuato, Universidad de Guanajuato

** Lic. Ingeniería en Sistemas Computacionales, División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato

*** Lic. en Artes Digitales, División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato

**** Departamento de Arte y Empresa, División de Ingenierías, Campus Irapuato-Salamanca, Universidad de Guanajuato.

j.bertaniramirez@ugto.mx[1], j.aguilerahuerta@ugto.mx[2],a.martinezledesma@ugto.mx[3] , m.stjohnclarkebideau@ugto.mx[4], le.dominguezhurtado@ugto.mx[5], wm.duransierra@ugto.mx[6] , uh.hernandez@ugto.mx[7] , ra.lizarragamorales@ugto.mx[8].

Resumen

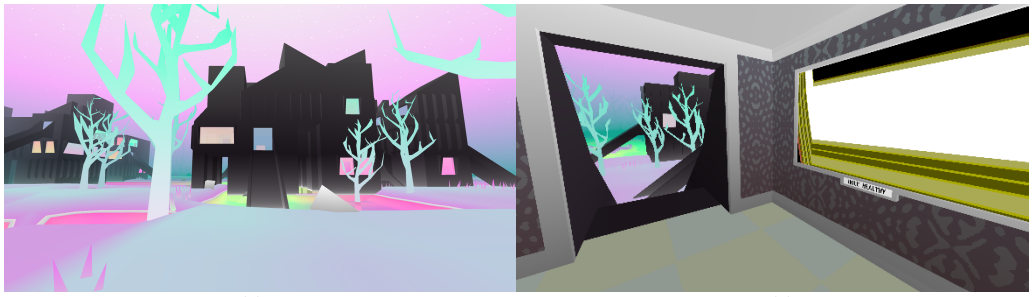
La generación procedural de contenido tiene capacidades para crear diferentes tipos de elementos (texturas, mapas, música, terrenos, vegetación, etc.) a partir de procedimientos precisos que buscan automatizar lo más posible la producción y el ahorro de costos en industrias como la de los videojuegos. Estas técnicas juegan un papel fundamental que nos abre las oportunidades de experimentar con diversas propuestas de contenido que se adapten a diferentes sentimientos, procesos cognitivos y lúdicos, a través de un buen análisis previo formal. Gracias a la exploración de áreas tan diversas (vegetación con sistemas-L, terrenos, mapas navegables, estéticos y didácticos), pudimos no solo generar 6 aplicaciones interactivas con aplicaciones de algoritmos procedurales, sino que esto nos permitió ampliar el panorama, combinar diversas áreas y llegar a conclusiones similares que analizan las cualidades y áreas de oportunidad en las diferentes aplicaciones que existen en la generación procedural de contenido.

Palabras clave: contenido procedural, humano-computadora, Lindenmayer, tortuga, celda, wave function collapse, colapso, tile, entropía, abrupto, mapa de altura, lúdico.

Introducción

La generación procedural de contenido (GPC) consiste en automatizar la mayor cantidad de procesos mediante el uso de algoritmos para generar contenido, en vez de crearlos de manera manual. Este contenido (texturas, sonido, mapas, vegetación, terrenos, etc.) es generado con entrada de información nula, limitada o indirecta de las personas que aplican estos métodos (Sánchez et al. 2019). Erróneamente se piensa que la GPC consiste en crear cosas exclusivamente a partir de la generación de números aleatorios, pero esto en aplicaciones reales genera algo que se conoce como ruido y encontrar algún patrón dentro de todos esos puntos es sumamente complejo o imposible, no hay ninguna aplicación práctica a partir de ello. En cambio, aplicando algoritmos que también usan números aleatorios y utilizando procedimientos controlados se puede generar contenido que se puede aplicar a muchos procesos, por ejemplo: un mapa de alturas con transiciones más suaves entre niveles.

La generación de terrenos son solo un área de aplicación. Algunos otros ejemplos los podemos encontrar en videojuegos como *Spore,* con generación de música que se adapta a las acciones del jugador y al entorno que lo rodea, en *Secret Habitat*, todo es generado de manera procedural, islas con edificios, obras de arte y sonidos. En la Figura 1, se muestran dos capturas de pantalla de como luce el juego *Secret Habitat,* en la imagen b, se muestra una pintura la cual es generada proceduralmente a partir del título de esta.



(a) (b)

Figura 1. Capturas de pantalla de Secret Habitat.

La industria de los videojuegos se enfrenta a desafíos donde las personas que consumen sus productos buscan entregas en los menores tiempos posibles con buena calidad y atención al detalle. Esto comprende el generar historias, mecánicas, ambientes, contextos, personajes, muchas cosas que en conjunto formen una experiencia con la que conecten y se sientan inmersas. De manera natural, son exigencias que aún con grandes equipos de trabajo, requieren más tiempo y, sobre todo, inversiones mayores que orillan a las empresas a correr riesgos menores en el contenido que quieran desarrollar. Estas problemáticas son las que la GPC trata de resolver, cualquier proceso que le permita a la industria reducir costos de producción, permitir delegar tareas más importantes al personal y por consecuencia, mayor libertad creativa, siempre serán bienvenidos.

A continuación, mencionaremos algunas áreas de interés para la GPC. La generación procedural de árboles en un ambiente es necesaria cuando dicho ambiente involucra modelar muchos árboles, por ejemplo, un bosque que sean de un mismo tipo, pero que sean lo suficientemente diferentes para mantenerlos lo más apegados a la realidad posible, donde no existen dos árboles exactamente iguales. Ocurre un proceso similar con la generación de terrenos que deben seguir o simular ciertas características geográficas como la erosión del suelo, mesetas, valles, montañas entre otros. Esto implica el manejo de alturas del terreno y las restricciones relacionadas con representaciones simbólicas a la realidad. En otra instancia, la generación de mapas enfrenta mayor dificultad cuando se busca generar una experiencia interactiva valiosa a través de conectar de manera lógica diferentes bloques constructivos y conservar la coherencia en el diseño. Para garantizar esto, es necesario conocer cuáles son las capacidades que tiene el jugador dentro del mundo que creamos y que tenga todas las capacidades de navegar libremente por donde queremos. Sintetizar estas mecánicas y comportamientos siempre se consideran antes de producir o implementar alguno de estos algoritmos.

En este proyecto se plantearon cinco enfoques diferentes que exploran el uso de la GPC en la producción de videojuegos que llamaremos de la siguiente manera a lo largo de este artículo:

1. Vegetación con Sistemas-L. Imitando y sintetizando la complejidad de la naturaleza.
2. Diamantes y terrenos. Controlando lo pseudoaleatorio desde la simplicidad.
3. Creación de escenarios. Encontrando el valor de lo que no se puede controlar.
4. Búsqueda del tesoro. Niveles desafiantes en masa y únicos.
5. Calabozos procedurales. Creando escenarios navegables.

El contenido de este artículo abarcará en la sección 2 la metodología general de cada una de las aplicaciones desarrolladas. En la sección 3 se presentarán los resultados alcanzados y, por último, en la sección 4 se presentarán las conclusiones generales del proyecto.

Metodología

A continuación, se presentarán estos proyectos en el mismo orden que se plantean en la introducción, la metodología que se siguió para el desarrollo de cada enfoque, se describirán los conceptos importantes para cada uno y se finalizará con los detalles de la implementación de los algoritmos.

Vegetación con Sistemas-L

Sistema *Lindenmayer* (o Sistema-L) es el nombre que se le da a una gramática que permite modelar un proceso de crecimiento de plantas, estás estructuras contienen los siguientes objetos:

- Un conjunto de caracteres, llamada *alfabeto.*
- Una cadena de caracteres, llamada *axioma.*
- Y un conjunto de *reglas de producción* las cuales tienen la forma

  Carácter → Cadena de caracteres

  nos indican una transformación del caracter a la cadena. La parte antes de la flecha se llama *predecesor* y lo que está después, *sucesor.*

Teniendo estos elementos, definimos una *cadena resultante* de un Sistema-L como toda cadena que se obtiene de aplicar n veces todas las reglas sobre el axioma. Por ejemplo, si tenemos las reglas

- A → AB
- B → A

y comenzamos con el axioma A, las primeras 5 cadenas resultantes son las siguientes:

1. A
2. AB
3. ABA
4. ABAAB
5. ABAABABA.

Notemos que existe cierta regularidad en las cadenas obtenidas y es por esta misma razón que las podemos utilizar para simular estructuras biológicas. Para lograr este cometido vamos a usar algo conocido como gráficos Tortuga (Prusinkiewicz et al., 2004). En estos se considera que existe una tortuga con un lápiz a la que se le dan instrucciones para moverse y cuando lo hace, dibuja su camino. En un espacio bidimensional la tortuga se representa por la tripleta (x,y,) donde (x,y) indica su *posición* en el plano e indica hacia donde está su *frente.* Dado un *tamaño de paso* **d** y un *ángulo de incremento,* la tortuga responderá a las siguientes instrucciones representadas con caracteres:

- F Se moverá hacia adelante **d** unidades. Es decir, su estado cambiará a (x',y',) donde x'=x+d() y y'=y+d() y además, se trazará una línea entre (x,y) y (x',y').
- f Se moverá hacia adelante **d** unidades sin dibujar una línea.
- + Girar a la izquierda un ángulo delta. Así, el siguiente estado de la tortuga será (x,y,+)
- - Girar a la derecha un ángulo delta. Así el siguiente estado será (x,y,-)

Además, podemos considerar que también existe una pila de estados de la tortuga donde los caracteres "[" y "]" agregan las siguientes instrucciones:

- [Agregamos el estado actual de la tortuga a la pila.
-] Sacamos un estado de la pila y lo volvemos el estado actual de la tortuga. No se traza ninguna línea.

Los Sistemas-L que incluyen estos caracteres son conocidos como *Sistemas Lindenmayer Encorchetados* y generar las estructuras que estos permiten dibujar es el primer objetivo de esta sección.

Sistemas Lindenmayer Encorchetados en Processing

Para conseguir generar las estructuras mencionadas lo primero que se requiere es solicitar al usuario tanto los elementos que definen al sistema *Lindenmayer* como los que definen el comportamiento de la tortuga. Es decir, solicitaremos el axioma, las reglas, el número de veces que aplicaremos las reglas, la orientación inicial de la tortuga y el ángulo. La posición inicial y el tamaño de paso no son necesarios por una razón que explicaremos más adelante. Además, para evitar algún problema, consideraremos "=" como el símbolo para separar los elementos de las reglas en lugar de la flecha.
Usando la librería G4P, se obtuvo la siguiente interfaz (Figura 2) de usuario que permite introducir todos los datos anteriormente expuestos.

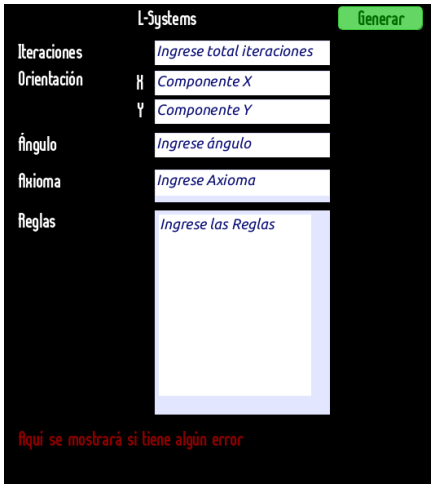


Figura 2. Interfaz de usuario generada en Processing usando G4P

Una vez solicitados y guardados los datos, debemos aplicar las reglas ingresadas la cantidad de veces indicada para obtener así la **cadena resultante** a dibujar. Para esto simplemente debemos seguir las reglas antes mencionadas y cada que leamos una "F" en la cadena, en lugar de dibujar la recta indicada, guardamos el punto inicial y final de esta. Una vez hayamos guardadas todas las rectas obtendremos de manera abstracta la figura por lo que podremos escalarla y trasladarla para así ponerla en cualquier lugar de la pantalla. Por esta razón evitamos pedir el tamaño de paso, ni la posición inicial. Teniendo estos valores como constante podremos controlar de mejor manera la posición y tamaño final del diagrama. A continuación, en la Figura 3 se muestra lo que podemos lograr siguiendo estas consideraciones tras implementarlas en Processing.

Figura 3. Dibujo generado (a la derecha) con el Sistema Lindenmayer ingresado a la izquierda

Sistemas Lindenmayer Paramétricos

En los *Sistemas Lindenmayer Paramétricos* ya no vamos a operar con cadenas de caracteres; ahora lo vamos a hacer con *cadenas de módulos.* Un *módulo* es simplemente un *carácter* con un *parámetro* asociado donde nuevamente los caracteres pertenecen a un *alfabeto* y los parámetros son números reales. Formalmente un módulo puede tener una cantidad ilimitada de parámetros, pero nos concentraremos en aquellos que solo tienen uno o ninguno. Así, si A es un carácter y x un parámetro arbitrario, un módulo de un solo parámetro se denota por **A(x)** y uno sin parámetros, solo por **A.** Por extensión, ahora el *axioma* será una secuencia de módulos, por ejemplo: F(1.1)AF(2.2)BCD(2.2).

Ahora, como vamos a operar con módulos, las reglas también deben cambiar. Estas ahora son de la forma *predecesor : condición → sucesor*. El **predecesor** es un módulo donde si este lleva parámetro se debe indicar que es arbitrario. En la **condición** debe ir alguna condición matemática que debe cumplir el parámetro del

predecesor para poder aplicar la regla o bien, puede omitirse para que siempre sea válida. Finalmente, el **sucesor** es una cadena de módulos en donde a cada módulo se le da un nuevo valor como parámetro (si lleva) o se le da una modificación del parámetro que provee el predecesor. Estas modificaciones se llevan a cabo mediante sumas, restas, multiplicaciones, divisiones y exponenciaciones. Así, las reglas de un Sistema-L paramétrico pueden verse así:

1. F(x) : x > 2 → F(x*3)F(x+1.2)A(x^0.5)
2. A(x) → A(x/3.14159)B

Una regla se aplica sobre un módulo si el carácter del módulo coincide con el del predecesor y el número de parámetros del módulo es el mismo que el del predecesor. Por ejemplo, si comenzamos con el axioma F(3) después de una iteración obtendremos la cadena F(9)F(4.2)A(1.732) pero si hubiéramos comenzado con F(1) no se hubiera aplicado ninguna regla. Por motivos de simplicidad, solo trabajaremos con Sistemas-L Paramétricos con reglas que no lleven condición y, como ya se dijo, con módulos de a lo mucho un solo parámetro.

De la misma manera que los Sistemas-L usuales, estos también pueden ser interpretados como instrucciones para gráficos tortuga. Para ello daremos su interpretación para una tortuga en el espacio tridimensional.

En tres dimensiones ya no será suficiente una tripleta y en su lugar usaremos 4 elementos para describir un estado de la tortuga: (p, H, L, U), donde p es un punto que indica su posición en el espacio y H, L y U son tres vectores unitarios que representan (respecto a la tortuga) dónde está su *frente*, su *izquierda* y su *arriba*, respectivamente, y además satisfacen que HL = U.

Las rotaciones ya no se pueden representar simplemente con una suma de ángulos y en su lugar usaremos la siguiente fórmula matricial para calcular la nueva orientación de la tortuga:

$$[H'\ L'\ U'] = [H\ L\ U]R$$

Donde R es cierta matriz de rotación de 3 x 3. Si queremos definir una rotación de ángulo alrededor de los vectores U, L y H usaremos las siguientes matrices respectivamente:

$$R_U(\alpha) = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R_L(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) \\ 0 & 1 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}$$

$$R_H(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

Como ahora tenemos parámetros, ya no es necesario usar dos símbolos para indicar la dirección de la rotación, pues podemos pasar el ángulo o su inverso aditivo para lograr este efecto. Es así que la tortuga obedecerá a las siguientes pocas reglas:

- F(a) Se moverá hacia adelante a unidades. Es decir, su estado cambiará a (p', H, L, U) donde p' = p + aH y además, se trazará una línea entre p y p'.
- f(a) Se moverá hacia delante a unidades sin trazar una línea.
- +(a) Girará sobre U un ángulo de a grados usando la matriz RU(a).
- &(a) Girará sobre L un ángulo de a grados usando la matriz RL(a).
- /(a) Girará sobre H un ángulo de a grados usando la matriz RH(a).

Pero esto no es todo, podemos extender estas reglas aún más y agregar las siguientes:

- !(a) Aumenta el grosor del trazo a a.
- $ Rota la tortuga para que L quede en posición horizontal. Esto se logra haciendo L = VH|VH| y U = HL, donde V es el vector (0,1,0).

VOLUMEN 16
XXVII Verano De la Ciencia
ISSN 2395-9797
www. jóvenesenlaciencia.ugto.mx

Y así, como su equivalente bidimensional, también podemos extender los Sistema-L paramétricos a aceptar corchetes, obteniendo los *Sistemas Lindenmayer Paramétricos Encorchetados* donde los corchetes fungen exactamente la misma función. Son las estructuras que estos sistemas generan el segundo objetivo de la sección.

Sistemas Lindenmayer Paramétricos Encorchetados en Unity

Lo primero que debemos hacer es solicitarle al usuario las partes que definen al Sistema *Lindenmayer* y lo necesario para que la tortuga pueda dibujar y como se hizo en la implementación en Processing, solamente necesitamos el axioma, las reglas, el número de veces que aplicaremos las reglas y la orientación inicial de la tortuga, mediante los vectores H, L y U. Estos parámetros son suficientes para generar los Sistemas-L Paramétricos encorchetados pero si queremos imitar estructuras biológicas más complejas también debemos tomar en cuenta el *Tropismo.* En biología, esta es la influencia del ambiente sobre el crecimiento de una planta pero aquí lo consideraremos como un vector (Prusinkiewicz et al., 2004). Si T es la dirección del tropismo, cada que avance la tortuga ajustaremos su orientación moviendo el vector H hacia el vector T un ángulo de = e|HT| donde e es la susceptibilidad de la planta a doblarse. Entonces, el usuario también debe ingresar estos dos valores para aplicar el tropismo.

Usando las herramientas de Unity para crear Interfaces Gráficas obtenemos el siguiente elemento que permite ingresar al usuario las variables anteriores.

Figura 4. Interfaz creada en Unity usando sus herramientas de Interfaz Gráfica

Notemos que tenemos un botón para indicar si queremos generar un árbol o no. Esto es porque todos estos parámetros nos permiten generar estructuras similares a árboles y para más realismo se decidió que al activar esta opción se use una textura de madera al momento de trazar el árbol y se agreguen hojas en donde haya un módulo con carácter A. Además de que el tropismo es exclusivo para estas estructuras.

Una vez tengamos los datos solo debemos aplicar las reglas la cantidad que se indique de veces y proceder a dibujar la estructura, solo que una vez más volveremos a guardar los trazos como punto inicial y final para poder trasladar la estructura final y asegurar que la cámara lo renderice por completo. Se decidió trazar las líneas con cilindros para aplicarles la textura de madera cuando fuera indicado y obtener así estructuras como las siguientes.

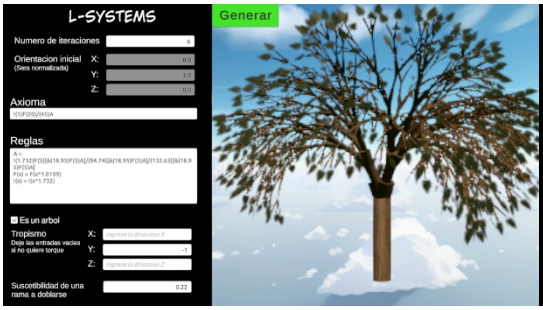


Figura 5. Árbol generado con un Sistema Lindenmayer paramétrico

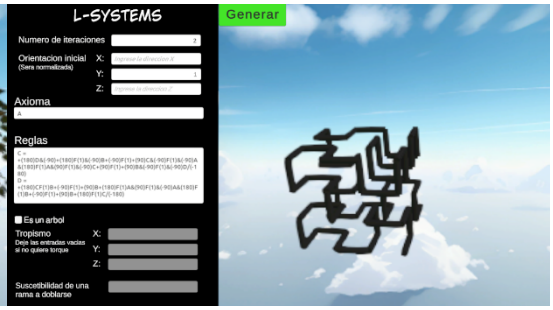


Figura 6. Aproximación simple a la curva de Hilbert

Diamantes y terrenos.

Para la implementación de esta aplicación partimos de una matriz de dos dimensiones con una longitud de *n x n* (la superficie debe ser cuadrada para garantizar mejores resultados). Exceptuando los valores de las 4 esquinas que delimitan el terreno y se generan de manera aleatoria, el resto de los valores de la matriz se calculan a partir del algoritmo, en los siguientes dos pasos:

a) **Paso del diamante:** Para cada cuadrado de la matriz, establece que el punto medio de ese cuadrado sea la media de los cuatro puntos de las esquinas más un valor aleatorio.
b) **Paso del cuadrado:** Para cada diamante de la matriz, establece que el punto medio de ese diamante sea la media de los cuatro puntos de las esquinas más un valor aleatorio.

En cada iteración, el rango aleatorio que se suma al promedio se divide a la mitad para garantizar una distribución donde a partir de una altura máxima en un punto del terreno, sus puntos vecinos vayan disminuyendo su altura poco a poco. En algunos casos, el paso del Cuadrado donde los puntos situados en los bordes de la matriz tendrán sólo tres valores adyacentes establecidos en algunos casos, hay varias maneras de solucionar este problema, pero en este proyecto lo solucionamos sacando la media de los valores que se encuentran adyacentes.

Rango de altura máximo

Buscando mejorar la experiencia de usuario, dentro del programa existe una función que permite fijar un rango automático de alturas según el tamaño del terreno. Esto permite que no se generen mapas muy pequeños con cambios tan drásticos entre alturas y le garantiza una experiencia controlada al usuario sin limitarle la libertad para que cree los entornos que desee.

Después de analizar y probar con diferentes terrenos y variables, se llegó a una conclusión con la relación entre el tamaño de alturas aleatorio y el tamaño del terreno; se representaba en una función continua a segmentos. Esta no es una interpretación absoluta, solo es una alternativa para que las personas no deban concentrarse en modificar valores un poco más confusos para generar terrenos de calidad. Además, este rango comprende también la parte negativa para generar terrenos de mayor calidad y no mantener relaciones planas de 0 a infinito.

Ajustes de parámetros para controlar la generación del terreno

Una vez tenemos todos los valores de la matriz con el correcto procedimiento, procedemos a crear el terreno en Unity, para esto necesitamos un modelo de un cubo preestablecido que será un cubo de tamaño 1x1x1 que instanciaremos y modificaremos su color y altura dependiendo del valor que se haya generado, formando así una cuadrícula de nuestro terreno.

Creación de escenarios

El algoritmo *Wave Function Collapse* (WFC) puede ser implementado de diversas maneras. La manera concebida por Maxim Gumin (2016) involucra una imagen de referencia que permite generar imágenes de

VOLUMEN 16
XXVII Verano De la Ciencia
ISSN 2395-9797
www. jóvenesenlaciencia.ugto.mx

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 7. 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 8. Cuadrícula con la celda central definida en agua

Vemos que, el pasto no puede colindar con el agua, el pasto deja de ser una posibilidad para las celdas colindantes y solamente se podría poner, agua 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 como en la Figura 5:

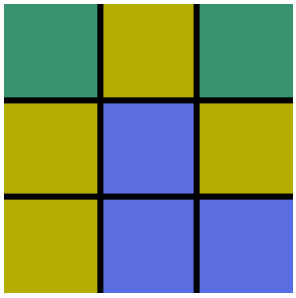


Figura 5. Cuadrícula con cada celda definida

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

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. 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. 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 anteriormente 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. Se crea una lista que almacenará las celdas afectadas, dato que cobrará sentido en los siguientes puntos.
    1. Se agrega la celda colapsada a la lista como su único elemento inicial.
    2. Se entra a un ciclo que itera cuantas veces como elementos haya en esa lista.

3. 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.
4. 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.

Búsqueda del tesoro

La teoría para esta aproximación es similar a la presentada previamente para la creación de escenarios. Sin embargo, como todos los contenidos procedurales, cada interpretación y adaptación es diferente y es un excelente ejemplo para demostrar la versatilidad de la GPC. La creación de este videojuego requirió la generación de un terreno procedimental en el que se escondería un tesoro en una casilla aleatoria. Para lograr esta parte procedural, se desarrollaron dos clases fundamentales: "Celda" y "Opciones". La clase "Celda" representa cada una de las celdas que conforman la cuadrícula, que, a su vez, constituye nuestro terreno. Por otro lado, la clase "Opciones" hace referencia a las diferentes posibilidades o "tiles" dentro de cada celda.

Antes de adentrarnos en los detalles del desarrollo, es imprescindible definir las reglas del videojuego. Para ello, es necesario realizar un análisis de los "tiles" y cómo se componen, tal como se puede observar en la Figura 8.

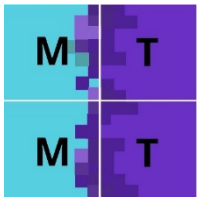


Figura 8. Muestra de un tile

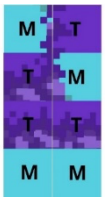


Figura 9. Diferentes orillas que puede tener un tile.

Figura 10. Reglas para nombrar tile

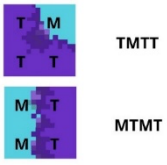


Figura 12. Tiles, TMTT, MTMT

Con esta información como base, podemos explicar con mayor detalle el proceso de implementación del videojuego "Buscando un diamante". Observamos que las "tiles" que utilizaremos están compuestas por dos

tipos: M y T, donde la M representa el Mar y la T la Tierra. A partir de esta observación, podemos deducir que cada borde del "tile" tiene 4 opciones posibles, las cuales serían: MT, TM, TT y MM.

La opción MT indica que el borde en cuestión tiene una transición de Mar a Tierra, mientras que TM representa una transición de Tierra a Mar. Por otro lado, TT indica que el borde está completamente compuesto por Tierra, y MM señala que es completamente Mar (Figura 9). En cuanto a la cuestión del orden y la etiquetación de cada "tile", se presenta en la Figura 10 para clarificar y estructurar visualmente las opciones de las orillas. En la Figura 11, se ha establecido un etiquetado para cada "tile". Las opciones de la orilla superior se representan como "AB", mientras que las opciones de la orilla inferior se denotan como "CD". A partir de estas opciones, es posible obtener la configuración para las orillas derecha e izquierda, donde la orilla derecha sería representada por "BD" y la izquierda por "AC". Con esta información, comprendemos que conocemos las orillas de cada "tile" y podemos compararlas para determinar si pueden unirse entre sí o no. Cada instancia de la clase "Celda" cuenta con un arreglo dinámico de instancias de la clase "Opciones", lo que nos permite agregar y eliminar opciones según sea necesario.

Para facilitar la representación del terreno y su disposición, se crea una cuadrícula utilizando una matriz bidimensional de instancias de la clase "Celda". En este caso, se ha establecido un tamaño de terreno de 10x10 donde cada celda contendrá sus respectivas opciones, representadas por los "tiles". Al tener la información estructurada de esta manera, se podrá realizar una evaluación y comparación sistemática entre las opciones de las celdas adyacentes para determinar si pueden encajar entre sí o si existen restricciones para su unión.

El algoritmo de generación procedimental para el videojuego "Buscando un diamante" se desarrolla siguiendo los siguientes pasos:

- Se realiza un análisis de todas las celdas para encontrar aquella que tenga la entropía más baja. La entropía hace referencia a la cantidad de opciones que tiene cada celda. En el inicio del proceso, todas las celdas tienen la misma entropía, por lo que se selecciona la última celda analizada que no tenga una entropía menor que la penúltima celda analizada.
- La celda seleccionada se colapsa, lo que significa que se elige una opción aleatoriamente de entre las disponibles y se le asigna ese valor.
- Cada opción tiene asociado un "tile" de los 14 posibles, y con ello, sus reglas. Cada "tile" cuenta con 4 orillas: superior, inferior, izquierda y derecha. Al colapsar una celda y asignarle un valor, sus reglas se propagan a las celdas vecinas. Cada celda posee un arreglo dinámico de opciones, lo que permite agregar o quitar opciones según corresponda. Al colapsar una celda, las celdas vecinas eliminan las opciones que no son compatibles con el valor asignado en cada orilla.
- Al quitar opciones, la entropía de las celdas disminuye, lo que hace que las celdas con menor cantidad de opciones vayan colapsando progresivamente. Este proceso evita posibles fallos en el algoritmo.

El proceso de análisis y colapso de celdas se repite en ciclos hasta que todo el terreno esté lleno y todas las celdas tengan valores asignados. En resumen, el algoritmo avanza iterativamente evaluando cada celda y seleccionando la de menor entropía para colapsarla y asignarle un valor. Luego, las reglas de ese valor se propagan a las celdas vecinas, lo que conlleva a eliminar opciones incompatibles y reduce la entropía en el proceso. El ciclo se repite hasta que todas las celdas tengan sus valores y el terreno esté completamente generado.

Calabozos procedurales

La generación de niveles de esta aproximación comienza con la creación de una ruta. Esta ruta sigue una serie de condiciones que le indican cuándo ir a la izquierda o a la derecha y cuando bajar, contemplando, de antemano, un margen que ponga un límite para la creación del terreno y contando con la distancia que tendrá una habitación de otra. La generación de niveles contempla diferentes pasos que serán descritos a continuación.

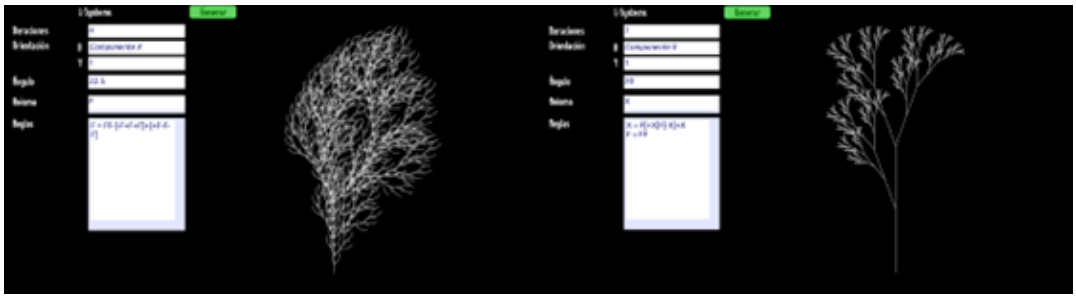
1. Generación de la maqueta. Para esto, es necesario comenzar a maquetar nuestra escena. En nuestro proyecto, por ejemplo, colocamos nuestros Puntos de Spawn en filas de 8x4 con una separación de 10 unidades cada uno.
2. Generación de la ruta. En el código, cuando se inicializa, necesita que el usuario coloque una “Posición inicial” en la que comience la ruta del nivel. cuando el código reconoce que existe una Posición Inicial, el código se encarga de inicializar una sala aleatoria, y de escoger un numero aleatorio para determinar la próxima posición.

3. Verificación de la ruta. El código utilizará la función para escoger un número aleatorio del uno al cinco y dependiendo de su elección, tomará un camino u otro. Para ser específicos estas son las direcciones que tomara:
    - Si “dirección” es igual a uno o dos la ruta irá a la derecha.
    - Si “dirección” es igual a tres o cuatro la ruta irá a la Izquierda.
    - Si “dirección” es igual a cinco la ruta irá hacia abajo.
4. Resolviendo contradicciones. Si en dado caso, por la asignación aleatoria de las salas, se coloca una sala que bloquee el camino de la ruta, el código eliminara y remplazara la sala con sala correcta.
5. La generación aleatoria de salas. Cuando la ruta acabe, aún sigue habiendo espacios vacíos en nuestra escena. Para esto, el código se encargará de rellenar esos espacios vacíos con salas aleatorias. Terminando así la generación procedural de nuestro nivel.

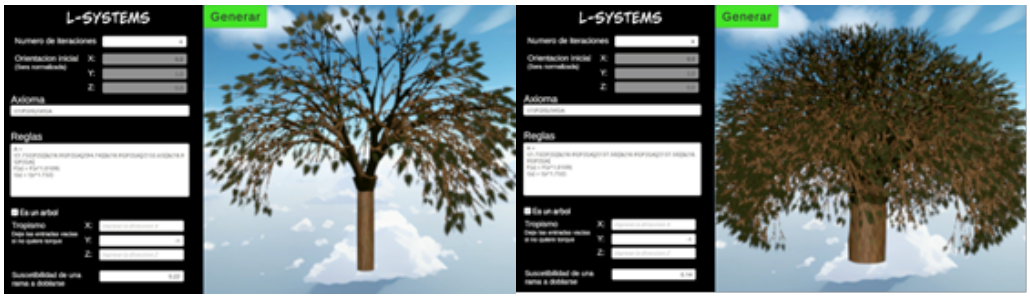
Resultados

Como resultados de este trabajo, se lograron 5 aplicaciones de diferentes métodos relacionados con la GPC. A continuación, se presentan algunas capturas de pantalla de cada una de ellas.

Vegetación con Sistemas-L.



Figuras 12-17. Estructuras similares a plantas generadas con Sistemas-L encorchetados.



Figuras 22-25. Estructuras parecidas a árboles generados por Sistemas-L paramétricos

Diamantes y terrenos.

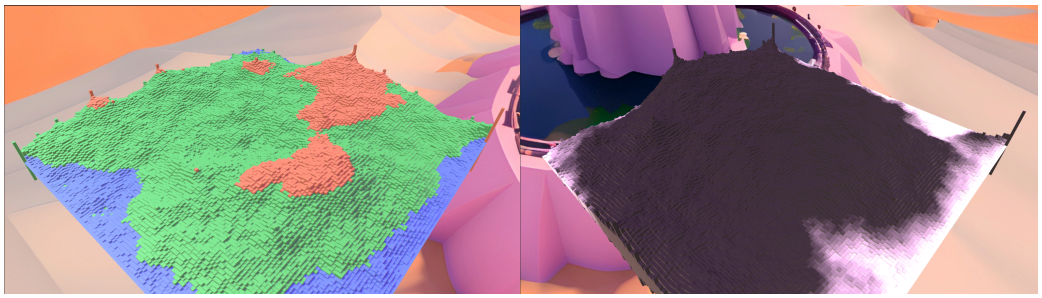


Figura 26. Generación de terreno (con materiales y sin materiales) de 129x129 y un rango aleatorio de -35 a 35.

Creación de escenarios.

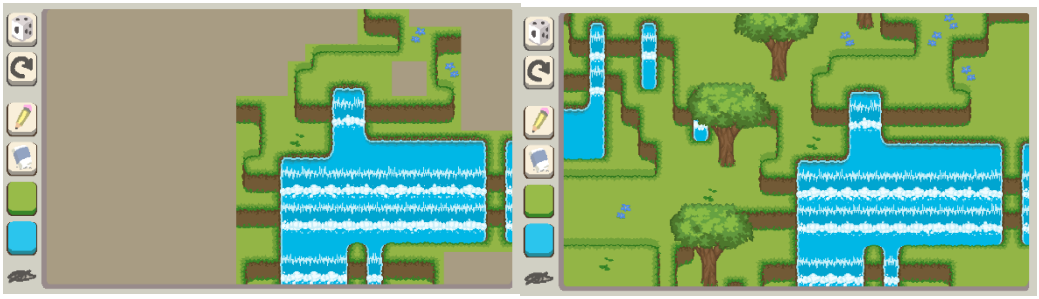


Figura 27. Captura del proceso de generación de un escenario y su resultado final

Búsqueda del tesoro.

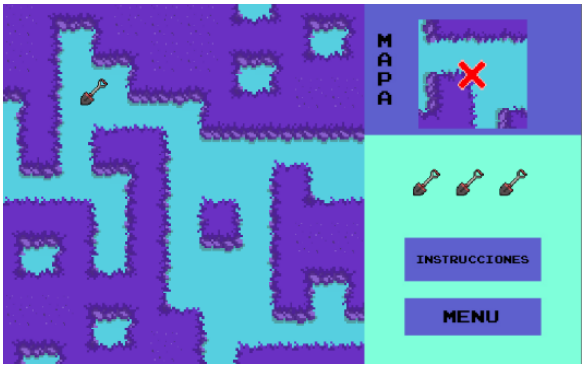


Figura 28. Vista principal del Videojuego Búsqueda del Tesoro

Calabozos procedurales.

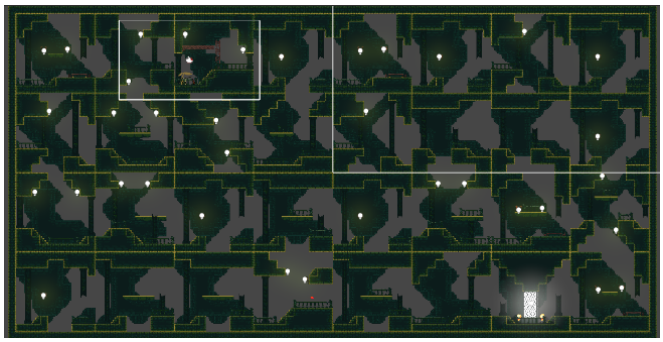


Figura 29. Escenario de calabozos completo en el videojuego creado.

Conclusiones

En este documento se presentaron 5 aplicaciones de diferentes técnicas de generación de contenido por procedimientos, cada una fue abordada desde diferentes perspectivas de diseño porque se tenían diferentes necesidades con la implementación de cada una. De manera resumida, con los sistemas-L se creó una herramienta para explorar las posibilidades de crear diferentes axiomas para resultados muy variados, desde árboles, aproximaciones de fractales, bioestructuras con parecidos a la naturaleza y muchas otras cosas. Para los terrenos y diamantes, ser conscientes del papel tan importante que juegan diversos parámetros para determinar la rugosidad, alturas, alteraciones en el terreno y nuevamente, dejar al usuario experimentar y probar generaciones nuevas. Por otro lado, generar terrenos que sean totalmente navegables según las capacidades que le atribuimos a las personas que juegan nuestros juegos.

Además de las posibilidades de navegar por el mapa, maneras de crear estos mapas explorando las decisiones que hay detrás de la aplicación de un algoritmo procedural como el wave function collapse, la manera en que se pueden alterar los parámetros de entrada para que tengas un mayor control de la estética, aportar a un buen sistema humano-computador estableciendo un buen vínculo con el jugador, de manera simultánea con el proyecto 4, la búsqueda del tesoro busca generar espacios didácticos a partir del mapa generado con el mismo método.

Comprender el estado del arte en base a una investigación de exploración, entender las limitaciones que siguen existiendo en el área y poco a poco adentrarse más en base a estos proyectos son excelentes iniciativas que en un futuro tal vez pueda permitirnos realizar contribuciones a problemas más específicos, muchas iniciativas de los algoritmos que generan contenido procedural van ligadas a la industria de los videojuegos y lo increíblemente exigente que es esta industria con su contenido y tiempo de realización.

Es muy probablemente que muchos de nuestros títulos favoritos nunca se hubieran explorado de no ser por las facilidades que otorga la generación procedural para crear iteraciones tan únicas en un determinado tema. Generar algo de manera procedural siempre te llevará primero a desarrollar un análisis preciso de aquello que quieres comprimir en el procedimiento, esta es una tarea que poco a poco se va perdiendo, la etapa de diseño siempre será el sustento de los mejores proyectos que se puedan ver y la GPC nos ayudó a recordar su valor, incluso en proyectos relacionados a otras cosas. Trabajar en una investigación de exploración es el primer paso para lograr contribuir a una problemática más real en un futuro, las motivaciones se alzaron a partir de leer otros trabajos, aplicaciones y beneficios.

Los videojuegos son un medio multimediático que siempre va a coexistir con otras disciplinas que le aportan una esencia única a la interacción humano-computadora, una razón más para detenerse a pensar en el diseño mucho antes de empezar a producir cualquier aplicación, seguir procesos y metodologías ordenadas para generar experiencia y conocimientos en otras áreas, la generación procedural resulta en un nexo muy interesante entre disciplinas computacionales, gráficas, estéticas, algorítmicas, matemáticas expuestas en una percepción de la realidad verosímil o fantástica.

Referencias

Cebollero, P.(2022).Sistemas de Sistema de generación procedural de niveles aplicado al género de videojuegos "Rogue-Like" [Grado en Ingeniería Informática, Universidad San Jorge Escuela de Arquitectura y Tecnología] Recuperado de: https://repositorio.usj.es/bitstream/123456789/862/1/Sistema%20de%20generaci%C3%B3n%20procedural%20de%20niveles.pdf

Fraile-Jurado, P. (2023). Geographical Aspects of Open-World Video Games. Games and Culture, 15554120231178871.

Fournier, A., Fussell, D., & Carpenter, L. (1982). Computer rendering of stochastic models. Communications of the ACM, 25(6), 371-384.

G. N. Yannakakis and J. Togelius, "Experience-Driven Procedural Content Generation," in IEEE Transactions on Affective Computing, vol. 2, no. 3, pp. 147-161, July-Sept. 2011, doi: 10.1109/T-AFFC.2011.6.

Kazemi,D. Spelunky Generator Lessons. (s.f.). Recuperado de: http://tinysubversions.com/spelunkyGen/

Prusinkiewicz, P. y Lindenmayer A. (2004) The Algorithmic Beauty of Plants. Springer. http://www.algorithmicbotany.org/papers/abop/abop.pdf

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

Sánchez Bouza, C., & Romero Pareja, Á. (2019). Creación de mundos mediante generación procedural en Unity.

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