Ciencia Latina Revista Científica Multidisciplinar
Noviembre-Diciembre, 2023, Volumen 7, Número 6
https://doi.org/10.37811/cl_rcm.v7i6.9305

Desarrollo de un Algoritmo Procedimental de Generación de Estructuras para su Implementación en el Desarrollo de un Videojuego 2D

Lenin Acosta Guevara[1]
rkb.lenin@gmail.com
https://orcid.org/0009-0000-6381-1693
Universidad Técnica de Ambato
Ambato, Ecuador

Félix Fernández Peña
fo.fernandez@uta.edu.ec
https://orcid.org/0000-0003-0834-3377
Universidad Técnica de Ambato
Ambato, Ecuador

RESUMEN

Los algoritmos de generacion procedural con frecuencia son complejos y computacionalmente costosos. Con el presente trabajo, se propone un algoritmo de baja complejidad basado en pilas para producir una matriz de valores correlacionados aplicable al desarrollo de mapas de videojuegos 2D. La matriz resultante se interpreta como un mapa de altura, y mediante el ajuste de hiperparámetros, se pueden controlar características como la rugosidad o la topologia del mapa. La generación de geometría dentro de la matriz está esencialmente condicionada por el criterio que se defina para almacenar la información en la pila, implicando que es posible obtener diversos resultados aplicables a más de un género de videojuegos únicamente cambiando la forma en que se almacenan los nuevos datos. Algunos de los resultados incluyen geometría fractal semejante al fractal de caja, geometría arbórea al estilo L-System y cuevas semejantes a las que se obtendría con autómata celular. Finalmente se realizó una implementación en Godot Engine en un equipo de bajos recursos para generar dos tipos de mapas demostrando la aplicabilidad del algoritmo que se propone.

Palabras clave: generacion procedural de contenido; algoritmos; videojuegos; estructuras de datos

[1] Autor principal
Correspondencia: rkb.lenin@gmail.com

Development of a Procedural Algorithm for Structure Generation for its Implementation in the Development of a 2D Video Game

ABSTRACT

Procedural generation algorithms are often complex and computationally costly. This work proposes a low-complexity stack-based algorithm to produce a matrix of correlated values applicable to the development of 2D video game maps. The resulting matrix is interpreted as a height map, by adjusting hyperparameters, characteristics such as roughness or topology of the map can be controlled. The generation of geometry within the matrix is essentially conditioned by the storage order criteria defined for storing information in the stack, implying that it is possible to obtain multiple results applicable to more than one genre of video games simply by changing the way new data is stored. Some of the results include fractal geometry similar to the box fractal, tree-like geometry in the style of L-System, and caves similar to those that would be obtained with cellular automata. Finally, an implementation was carried out in Godot Engine on a low-resource hardware to generate three types of maps, demonstrating the applicability of the proposed algorithm.

Keywords: procedural content generation; algorithms; videogames; data structures

Artículo recibido 17 noviembre 2023
Aceptado para publicación: 29 diciembre 2023

INTRODUCCIÓN

La generacion procedimental o conocida tambien como “generacion procedural” es usada para crear contenido de forma algorítmica, constituida por técnicas que reducen la intervención del humano (Dahren, 2021). Las técnicas de generacion procedural, que evolucionan a la par de los video juegos, usan diversos algoritmos de distinta complejidad para generar diferentes componentes (Melotti, 2019). La industria del entretenimiento utiliza la generación procedural para crear contenido dinamico y cubrir la demanda (Viana, 2021). Dado que posee los recursos, invierte en la investigación y desarrollo de algoritmos procedurales, que lamentablemente son de uso exclusivo (Amato, 2017) (Hollstrand, 2020). Los desarrolladores independientes o pequeños equipos, con recursos limitados, buscan alternativas rápidas y sencillas para generar más contenido y mejorar su flujo de trabajo (Hollstrand, 2020).

Las técnicas de generación procedural basadas en algoritmos de inteligencia artificial suelen enfocarse en resultados predecibles y controlables (Abraham, 2023) (Brown, 2018) (Zakaria, 2023) (Moreno-Armendariz, 2022). Sin embargo, estos algoritmos frecuentemente son de alta complejidad y por consecuencia computacionalmente costosos (Hollstrand, 2020).

Por este motivo, existe la necesidad de crear algoritmos con menor costo computacional con la capacidad de generar mapas para videojuegos 2D en equipos de bajos recursos. El presente trabajo propone una revision de la literatura para analizar soluciones vigentes y explorar nuevas alternativas. De esta forma encontramos investigaciones que se han centrado en innovadoras propuestas menos complejas para la generación procedural (Lipinski, 2019) (Johnson, 2010) (Macedo, 2017) (Putra, 2023) (Linden, 2014). Algunas soluciones incluyen el uso de grafos (Lipinski, 2019) y de modelos computacionales como autómata celular (Johnson, 2010) y fractales (Putra, 2023), para la generación de la estructura del mapa del videojuego en cuestión.

La geometría fractal, reconocida por sus formas intrincadas y visualmente atractivas, es estudiada y aplicada para resolver problemáticas de distinta índole en diferentes campos. Su aplicación va desde el ámbito artístico (Alghar, 2023) , pasando por su aplicación en electrónica (Li, 2024), ciberseguridad (Ayubi, 2020) (Chang, 2023), hasta la generación procedural en videojuegos (Putra, 2023) (Linden, 2014). Lipinski et al, resalta que para lograr el resultado que espera, necesita generar un grafo base y procesarlo para conectarlo y corregir irregularidades indeseadas (Lipinski, 2019). Jonhson indica que

requiere de al menos cuatro iteraciones para lograr el resultado deseado (Johnson, 2010). Finalmente Putra et al, quienes proponen el uso de L-system, se ve involucrado en el uso de funciones recursivas, por lo que depende de una cuidadosa optimización para evitar costos computacionales innecesarios (Putra, 2023).

Al comprobar la complejidad algorítmica de las propuestas de otros autores, el objetivo que se propone con el presente trabajo es desarrollar un algoritmo eficiente para generar mapas 2D para videojuegos, capaz de funcionar en equipos con recursos limitados. Se propone que el algoritmo se base en la estructura de datos pila, para minimizar el costo computacional en el proceso generativo. Mediante la configuración de hiperparámetros, el algoritmo podrá generar una variedad de configuraciones geométricas, incluyendo geometría fractal.

METODOLOGÍA

Se emplea un enfoque cualitativo en una investigación exploratoria con el propósito de analizar la literatura para comprender la efectividad de los algoritmos propuestos y explorar nuevas alternativas para generar mapas en videojuegos 2D. Esta metodología implica una combinación de diseños observacionales y experimentales, lo que permite una evaluación detallada del rendimiento y la versatilidad del algoritmo. Además, se realiza una revisión documental para contextualizar el aporte de estudios previos relacionados con la generación de mapas y algoritmos procedurales. Este enfoque integral pretende no sólo analizar las capacidades de los algoritmos, sino también situar la presente investigación dentro de la evolución y estado actual del campo.

Para el desarrollo de la propuesta se emplea un enfoque cuantitativo experimental, fundamentado en la implementación del algoritmo propuesto y la evaluación experimental de sus capacidades en la generación de mapas 2D concretos.

RESULTADOS Y DISCUSIÓN

En esta investigación se ha definido un algoritmo (Algoritmo 1) que distribuye valores de manera correlacionada en un espacio bidimensional a partir de un valor semilla, generando una matriz de valores útiles para generar mapas de videojuegos 2D. Se ha evaluado la utilidad de las pilas y cómo el almacenamiento de datos en ellas puede generar entropía y producir diversos resultados.

En la Tabla 1 se aprecia la nomenclatura utilizada para referirse a los hiperparametros del algoritmo y sus funciones.

Tabla 1. Nomenclatura usada para referirse a los hiperparámetros.

Simbolo	Descripcion	Dominio
m	Numero de filas.	$m \in Z+; m > 1$
n	Numero de columnas.	$n \in Z+; n > 1$
i	Posicion en fila.	$i \in Z+; 0 \leq i < m$
j	Posicion en columna.	$j \in Z+; 0 \leq j < n$
ε	Valor numerico asosciado con δ o α.	$\varepsilon \in R+$
α	Espacio bidimensional de m×n.	
δ	Conjunto compuesto por ε,i,j.	
κ	Umbral que ε puede alcanzar.	$\kappa \in R; \kappa \geq 0$
λ	Umbral de distancia entre dos valores entre 0 y 1.	$\lambda \in R; 0 < \lambda \leq 1$
ν	Conjunto compuesto por δ vecinos.	
γ	Valor aleatorio.	$\gamma \in R; 0 \leq \gamma \leq 1$
ρ	Valor deterministico.	$\rho \in R; 0 \leq \rho \leq 1$
θ	Umbral de distancia entre dos ε.	$\theta \in R+$

Algoritmo 1. Generación de matriz con valores correlacionados

Algorithm 1 Main

$$\alpha = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}_{m \times n}$$

$\delta \leftarrow [\epsilon, i, j]$
$stack \leftarrow []$
$stack.append(\delta)$
while $stack$ has elements **do**
  $\delta \leftarrow stack.pop()$
  $\delta_\epsilon \leftarrow Correlation(\delta_\epsilon)$
  if $\delta_\epsilon \geq \kappa$ **then**
    if $0 < \delta_i < m - 1$ **then**
      if $0 < \delta_j < n - 1$ **then**
        $\nu \leftarrow Neighborhood(\delta)$
        $LocalRules(\nu)$
        $Insertion(\nu)$
      end if
    end if
  end if
end while

El Algoritmo 1 esta compuesto por distintas funciones donde se consideran los siguientes aspectos:

Correlación de valores: Aqui se describe un proceso en el que los valores en α son determinados por

una función aplicada a ε. Dicha función puede ser una progresión aritmética simple de la forma ε ← ε - C, siendo C la distancia de la progresion y por tanto el umbral para pasar entre diferentes valores, de esta forma obtenemos una manera simple y eficiente de crear y clasificar valores. Adicionalmente, si se define un criterio en el cual se le otorgue a ε una probabilidad ρ de restaurar su valores inicial se obtiene un α con una mayor cantidad de valores altos y bajos interactuando entre sí, lo que resulta en la generación eficiente de un sistema de cuevas pues estas se generan sin la necesidad de mas de una sola iteracion en α ademas de no requerir informacion adicional de su entorno local para crear la topologia deseada. Se define las funciones: Función 1 y

Función ***2*** aplicando ls comportamientos descritos.

Función 1. Correlación lograda con una progresión aritmética simple.

Function 1 Simple Correlation

$\epsilon \leftarrow \epsilon - C$
$return \rightarrow \epsilon$

Función 2. Correlación estocástica incorporando la función de correlación simple.

Function 2 Stochastic Correlation

$diff \leftarrow |\kappa - \epsilon|$
if $diff <= \theta$ **then**
  if $\gamma <= \rho$ **then**
    $\epsilon \leftarrow initial\_\epsilon$
  end if
else
  $\epsilon \leftarrow SimpleCorrelation(\epsilon)$
end if
$return \rightarrow \epsilon$

Vecindad y reglas locales: La estrategia escogida para considerar la vecindad puede afectar la dinámica general del proceso de generación. Asimismo, la implementación de reglas locales permite distinguir qué valores de la matriz son actualizados y que vecinos son ingresados a la pila. La vencidad se refiere a los espacios contiguos en α respecto a una posición i,j. Usando la vecindad de Von Neumann (Chen, 2005), los vecinos inmediatos están posicionados en las cuatro direcciones: arriba, abajo, izquierda y derecha. Definimos la Función 3 para determinar la vecindad de un δ dado. Se define reglas locales en Función 4 para determinar que valores en ε de los vecinos son ingresados

en α y que vecinos en ν serán posteriormente ingresados a la pila.

Función 3. Obtención de la vecindad de v.

Function 3 Neighborhood

$\nu = \{\ up;\ right;\ down;\ left\ \} :$
$up = [\delta_\epsilon; \delta_i - 1; \delta_j]$
$right = [\delta_\epsilon; \delta_i; \delta_j + 1]$
$down = [\delta_\epsilon; \delta_i + 1; \delta_j]$
$left = [\delta_\epsilon; \delta_i; \delta_j - 1]$
$return \rightarrow \nu$

Función 4. Implementación de reglas locales.

Function 4 Local Rules

$\epsilon \leftarrow neighbor_\epsilon$
$i \leftarrow neighbor_i$
$j \leftarrow neighbor_j$
$value \leftarrow \alpha_{i=i;j=j}$
if $value == 0$ **then**
    $\alpha_{i=i;j=j} \leftarrow \epsilon$
else
    $sum \leftarrow \epsilon + value$
    $\epsilon \leftarrow \epsilon / sum$
    $value \leftarrow value / sum$
    $diff \leftarrow |\epsilon - value|$
    if $diff \geq \lambda$ **then**
        $\alpha_{i=i;j=j} \leftarrow \epsilon \times (1 - diff)$
    else
        $\nu.delete(neighbor)$
    end if
end if

Almacenamiento de los datos: La forma en que se ingresa el dato para ser almacenado tiene un papel relevante. En función de esta se obtienen diversas formas geométricas en la matriz resultante. Esto implica que la elección de la estrategia de almacenamiento es un aspecto crítico que puede afectar significativamente los resultados obtenidos. Análogamente a un axioma en L-System (Putra, 2023), se seleccionan letras del alfabeto que representarán a cada uno de los vecinos: A, B, C, D ← arriba, derecha, abajo, izquierda. Las configuraciones de almacenamiento determina el orden en que los datos seran ingresados en la pila. Las funciones de almacenamiento que se definen toman como parametro ν y separan sus valores bajo el supuesto de que se encuentren en el orden A, B, C, D, para ingresarlos según la configuracion que se defina. Establecemos dos primeras configuraciones de almacenamiento, a partir de las cuales se construirán más configuraciones y en donde los elementos serán ingresados de la siguiente forma:

a) A → B → C → D. Presenta un patrón horario.

b) C → B → D → A. Definido con un patrón alternativo para introducir variedad.

Si en cualquiera de las configuraciones iniciamos la secuencia en un punto distinto, manteniendo su patrón, obtenemos otras tres configuraciones adicionales por cada uno, lo que resulta en ocho criterios base. Cada criterio se usa en la

Función 5 según se requiera. Las configuraciones resultantes son las siguientes, primeros 4 con orden horario y 4 con orden alternativo:

1) BAc ← (B → C → D → A).
2) BBc ← (C → D → A → B).
3) BCc ← (D → A → B → C).
4) BDc ← (A → B → C → D).
5) BA ← (C → B → D → A).
6) BB ← (D → A → C → B).
7) BC ← (A → D → B → C).
8) BD ← (B → A → C → D).

Establecemos más criterios, los cuales se constituyen de la secuencia de los 8 base presentados, por lo que nos referimos a ellos como configuraciones secuenciales. Logramos 4 configuraciones secuenciales en base a las configuraciones con patrón alternativo:

9) SBA ← (BC → BA → BD → BB).
10) SBB ← (BB → BC → BA → BD).
11) SBC ← (BD → BB → BC → BA).
12) SBD ← (BA → BD → BB → BC).

Logramos también 2 secuenciales con base en las configuraciones de patrón horario:

13) SBAc ← (BAc → BCc → BBc → BDc).
14) SBBc ← (BAc → BBc → BCc → BDc).

También fue posible lograr un secuencial constituido por otros secuenciales:

15) SSA ← (SBC → SBA → SBD → SBB).

Sin embargo, estos son únicamente algunos de los criterios con configuraciones estables más notables que se encontraron a través del proceso de experimentación que se llevó a cabo. Más configuraciones pueden ser halladas mediante la experimentación o definiendo criterios basados en los que ya se han

propuesto.

Función 5. Almacenamiento de los datos en pila.

Function 5 Storage

$A, B, C, D \leftarrow \nu$
$BA \rightarrow C, B, D, A \triangleright BA$ Should be replaced by your own criteria
$\nu \leftarrow BA$
$stack.push(\nu)$

El Algoritmo 1 de complejidad O(m × n), dado que en el peor de los casos todas las celdas de la matriz sean visitadas, produce ruido que es ajustable mediante la configuración de sus hiperparametros lo que resulta en diferentes tipos de formas. Dicha matriz puede ser representada como un mapa de altura, asignando un color a cada celda en función de su valor, como se observa en la Figura 1

Figura 1. ruido generado con el algoritmo propuesto.

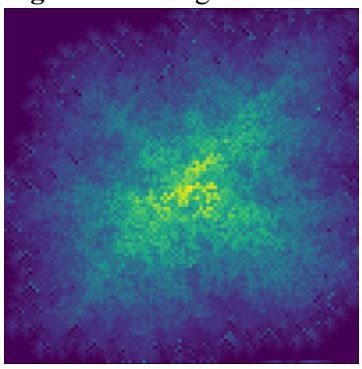
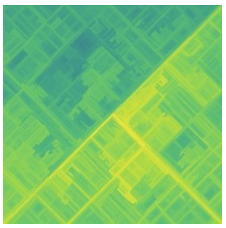
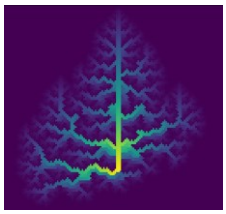
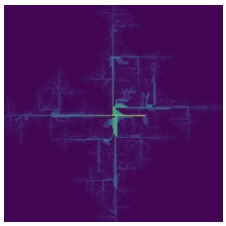
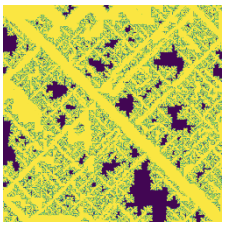
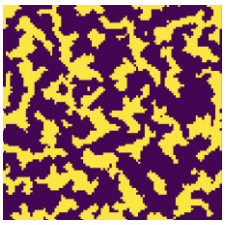


Tabla 2. Tipos de generaciones logradas mediante el ajuste de hiperparámetros.

Generaciones obtenidas con el algoritmo propuesto				
				
Generación del tipo corredor visto como mapa de altura.	Generación arbórea vista como mapa de altura.	Generación mixta entre arbórea y corredores vista como mapa de altura.	Generación de tipo corredor con valores clasificados usando la distancia C	Generacion de cuevas estilo autómata celular con valores clasificados.

Entre algunas de las generaciones posibles están las de tipo cueva y corredor, tal y como se aprecia en la Tabla 2. El tipo de generación corredor puede resultar útil tal y como explica Dahlskog, para conectar diferentes espacios o habitaciones, siendo incluso componente principal en algunos juegos (Dahlskog, 2015).

Dado que diseñamos nuestra propuesta pensando en su uso en equipos de recursos limitados, para generar los mapas no utilizamos aplicaciones exigentes como Unity o Unreal Engine que demandan equipos con altas prestaciones para poder usarlas sin problemas. En su lugar, optamos por Godot Engine debido a su ligereza y eficiencia. Godot Engine es un motor de juegos ligero y potente, ideal para equipos con recursos limitados.

Figura 2. Mapa de cuevas renderizado con Godot Engine.

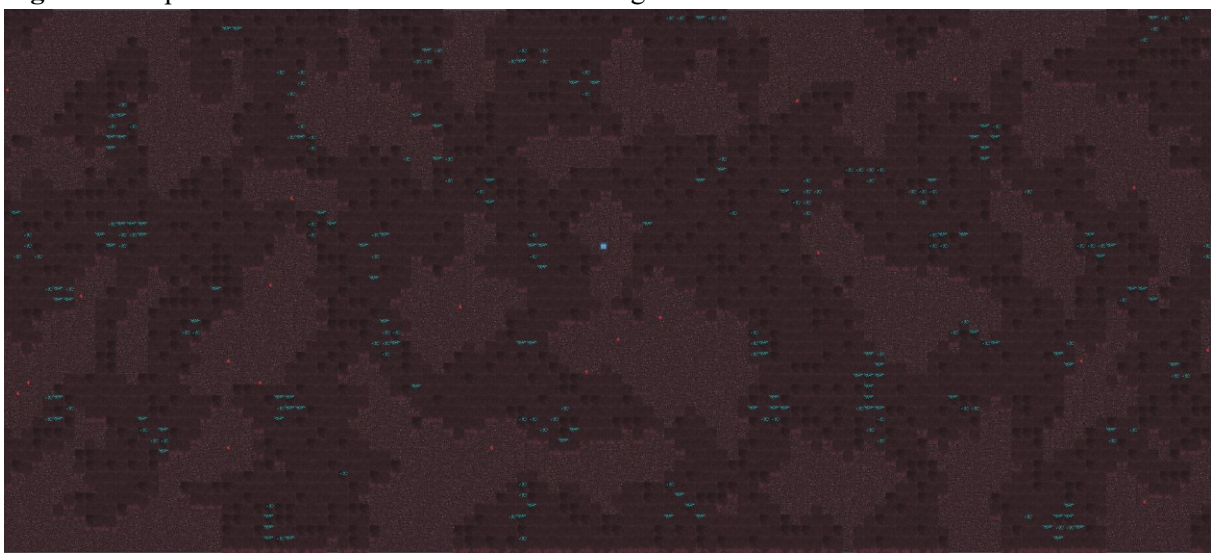


Figura 3. Mapa de corredores renderizado en Godot Engine.

En Figura 2 y Figura 3 se muestran los mapas renderizados con Godot Engine mediante la implementacion de nuestro algoritmo. Creamos dos ejemplares usando la generación de cuevas y corredores, respectivamente. El mapa que implementa corredores incluye vegetación, cofres, pilares,

baldosas y una fuente. Cada elemento tiene un rango definido que controla su ubicación y una probabilidad de aparición dentro de ese rango. Los cofres se colocan al final del camino, la vegetación se sitúa de manera que no interfiere con la ruta del jugador, las baldosas definen el área donde el jugador puede moverse, los pilares representan elementos que aparecen a medida que el jugador explora, y la fuente es el punto de origen para el jugador.

El mapa de corredores incluye elementos como vegetación, cofres, pilares, baldosas y una fuente. Para colocar y renderizar cada uno, se le asigna un rango y probabilidad de aparición. Los cofres se ubican al final del camino, la vegetación se coloca sin obstruir el pasaje del jugador, las baldosas definen el área de donde puede transitar el jugador, los pilares surgen a medida que el jugador explora, y la fuente es el punto de inicio del jugador.

El mapa de cuevas está compuesto de losetas de rocas y diamantes. Los jugadores pueden transitar por el espacio donde están las losetas de tierra. Los diamantes aparecen en las rocas y no en el camino, y las cerezas aparecen en el camino para que los jugadores las encuentren.

Hemos demostrado que es viable implementar y generar mapas coherentes de videojuegos 2D en equipos con recursos limitados. También hemos evidenciado la utilidad de generar un mapa de altura. Para ubicar diferentes objetos en el mapa, simplemente se les asigna un rango de los generados en la matriz. Esto nos permite controlar si los objetos pueden superponerse, encontrarse o aparecer en ubicaciones específicas.

CONCLUSIONES

Este trabajo presenta un algoritmo de baja complejidad algorítmica para generar mapas de videojuegos 2D en equipos de recursos limitados. El algoritmo permite el ajuste de hiperparámetros para crear valores de matriz correlacionados con diferentes topologías y distintos niveles de entropía para un mismo tipo de resultado. Al modificar la función de correlación y el orden de almacenamiento de los valores, se pueden obtener resultados diversos, incluyendo formas fractales. El algoritmo también permite la segmentación y definición de rangos para la colocación de elementos en el videojuego. Hemos demostrado que es posible generar mapas coherentes en equipos con recursos limitados y controlar la ubicación de los objetos en el mapa.

REFERENCIAS BIBLIOGRAFICAS

Abraham, F. a. (2023). Utilizing generative adversarial networks for stable structure generation in angry birds. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, (pp. 2-12).

Alghar, M. Z. (2023). Ethnomathematics: The exploration of fractal geometry in Tian Ti Pagoda using the Lindenmayer system. Alifmatika: Jurnal Pendidikan dan Pembelajaran Matematika, 57-69.

Amato, A. (2017). Procedural content generation in the game industry. Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation, 15-25.

Ayubi, P. a. (2020). Deterministic chaos game: a new fractal based pseudo-random number generator and its cryptographic application. Journal of Information Security and Applications.

Brown, J. A. (2018). Levels for hotline miami 2: Wrong number using procedural content generations. Computers, 22.

Chang, H. a. (2023). Research on Image Encryption Based on Fractional Seed Chaos Generator and Fractal Theory. Fractal and Fractional.

Chen, R.-J. a.-L. (2005). Data encryption using non-uniform 2-D von neumann cellular automata. 2005 9th International Workshop on Cellular Neural Networks and Their Applications, (pp. 77-80).

Dahlskog, S. a. (2015). Patterns, Dungeons and Generators.

Dahren, M. (2021). The usage of PCG techniques within different game genres.

Hollstrand, P. (2020). Supporting Pre-Production in Game Development: Process Mapping and Principles for a Procedural Prototyping Tool.

Johnson, L. a. (2010). Cellular automata for real-time generation of infinite cave levels. PCGames@FDG.

Li, R. a. (2024). Enhanced cooling performance of stacked chips by structural modification for fractal micro-protrusions. Applied Thermal Engineering.

Linden, R. V. (2014). Procedural Generation of Dungeons. IEEE Transactions on Computational Intelligence and AI in Games.

Lipinski, B. a. (2019). {Level graph-incremental procedural generation of indoor levels using minimum spanning trees. 2019 IEEE Conference on Games (CoG), (pp. 1-7).

Macedo, Y. P. (2017). Improving procedural 2D map Generation based on multi-layered cellular automata and Hilbert curves. 2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), (pp. 116-125).

Melotti, A. S. (2019). Evolving Roguelike Dungeons With Deluged Novelty Search Local Competition. IEEE Transactions on Games.

Moreno-Armendariz, M. A.-L. (2022). IORand: A Procedural Videogame Level Generator Based on a Hybrid PCG Algorithm. Applied Sciences, 3792.

Putra, P. A. (2023). Procedural 2D Dungeon Generation Using Binary Space Partition Algorithm And L-Systems. 2023 International Conference on Computer, Control, Informatics and its Applications (IC3INA), (pp. 365-369).

Toy, M. a. (1980). Rogue. Comput. Sci. Res. Group UC Berkeley.

Viana, B. M. (2021). Procedural Dungeon Generation: A Survey. Journal on Interactive Systems.

Zakaria, Y. a. (2023). Start small: Training controllable game level generators without training data by learning at multiple sizes. Alexandria Engineering Journal, 479-494.