Mapas Generados de Manera Procedural Mediante Geometría Modular en Unity

Adrian Abad
Carrera de Ing. en Realidad Virtual y Videojuegos
Universidad Católica de Cuenca
Cuenca, Ecuador
aecu12@gmail.com

Dante Gaibor
Carrera de Ing. en Realidad Virtual y Videojuegos
Universidad Católica de Cuenca
Cuenca, Ecuador
gaibordante@gmail.com

***Abstract*—En el área de desarrollo de videojuegos, la generación de mapas sufre una falta de herramientas para realizarlo de manera ágil y simple, causando un aumento de costos en la producción de un videojuego. La presente investigación tiene el objetivo de desarrollar un método de generación procedural de mapas para el motor de videojuegos de Unity en proyectos de 3D, utilizando la geometría modular para su creación. Se declararon 3 categorías para la creación del mapa: triangulares, cuadrangulares, y polígonos de 5 o más lados. Con cada categoría se utilizaron distintas fórmulas para generar correctamente los mapas mediante la geometría modular, además de prevenir un procesamiento excesivo al intentar utilizar una formula general para todas las figuras. Los resultados fueron mapas creados en la forma del mismo polígono de la cual están hechas las baldosas singulares que conforman el mapa, creando un nuevo estilo de generación de mapas.**

Keywords—Unity, geometría modular, mapas, videojuegos, juegos de estrategia

I. INTRODUCCIÓN

El desarrollo de videojuegos es la unión de varias destrezas para formar un producto final que apela a una gran audiencia (Shubin & Kugurakova, 2024), con habilidades como: la animación, arquitectura, administración de negocios, ingeniería, las matemáticas, la música y las artes visuales. Unidos, estos logran el diseño de un juego de calidad (Schell, 2008), además de satisfacer las necesidades y deseos del público objetivo.

Sin importar de que juego se habla, hay elementos claves que parecen intuitivos, pero a la vez, son las mecánicas las cuales definen el input del jugador y su reacción al entorno (Fabricatore, 2007). Por esta razón, estos componentes son considerados para armar el juego a pesar de su sencillez (Zubek, 2020). Se emplea un motor de juego (*game engine*), para poder implementar estas mecánicas esenciales (Politowski et al., 2020) sin tener que escribir el código en el motor, además de modificar propiedades del videojuego de manera simple. (Rabin S., 2010).

Esta es la razón principal por la cual es muy demandado la generación de mapas con formas geométricas, ya que permite que jugadores y NPCs (*non-playable characters* / personajes no jugables) puedan adaptarse fácilmente a cumplir distintos objetivos siguiendo pasos definidos. (Kim & Sung, 2024). Los patrones en figuras permiten un mejor almacenamiento de datos, que sirve para su optimización (Sahr, 2011). Sin embargo, realizar estos mapas usando código requiere grandes cantidades de procesamiento, principalmente para resolver ecuaciones matemáticas complejas y mantener una base de datos (Sharma et al., 2024). Como solución a este problema, se propuso generar el mapa de manera procedural con un diseño modular, que reduce la complejidad del diseño (Lai & Hu, 2025) mediante la segmentación del gran proceso en procesos más pequeños y simples de realizar (Kaźmierczak et al., 2024). Esto previene la necesidad de dependencias. (Mohammed Abdul et al., 2025).

II. ESTADO DE ARTE

Unity 3D es uno de los motores de videojuegos más usados en la industria (Haas, 2014), sobre todo en desarrolladores independientes, dado que el 45% de este sector trabaja con Unity (Foxman, 2019). Sin embargo, que sea el motor más usado no significa que sea perfecto. Existen funciones o herramientas que son de difícil accesibilidad para el desarrollador (Borrelli et al., 2020), pues el videojuego es tan diferente entre sí que el motor no puede abarcar todas las herramientas necesarias para los diferentes tipos de géneros (Gordo, 2020).

Una de estas herramientas esenciales que el motor no nos brinda es el mapa en la cual el juego se desarrolla, su importancia radica en brindar una vista estética al jugador y dar otro nivel de interacción e inmersión a la experiencia (Wang et al., 2024). El mapa en los videojuegos tiene diferentes funciones, una de ellas es localizar al jugador en el mundo (Khan & Rahman, 2018). Otra función, específica para videojuegos de estrategia, tanto como por turnos como en tiempo real, es servir para movilizar y colocar las unidades jugables en un mapa que se divide en casillas (Mukherjee, 2015). Los videojuegos de estrategia en tiempo real suponen un desafío para los estudios de videojuegos por la complejidad del tamaño del proyecto (Walfisz et al., 2006).

Por mucho tiempo no se actualizaron los instrumentos para crear mapas a la par de la demanda, y así brindar una mejor calidad y variedad en los instrumentos. (Smelik et al., 2009). Los fondos se destinaban a crear mejores mapas con los métodos antiguos cuando podían ser utilizados para los otros aspectos del juego como mecánicas innovadoras (Kelly G., 2006). Por esto se empezó a desarrollar la generación procedural de contenido (*procedural content generation, PCG)*, para crear diferentes aspectos del juego de manera algorítmica (Van Der Linden et al., 2014). Además, PCG sirve para adaptar los aspectos de la jugabilidad en tiempo real mientras corre el juego. (Hartsook et al., 2011).

La generación del mapa sigue siendo un reto debido a su dificultad, a pesar de las herramientas que ya se tenían, muchos estudios lo dejan como último pendiente debido a la complejidad que existe para poder crear estos mapas, como es el caso de *"Cosmic Crusade"* (Yallico et al., 2025), donde el jugador se sienta relacionado con el mundo y la exploración (Lammes & Wilmott, 2018). Por último, aunque el desarrollador haya diseñado el mapa siempre existe el reto de que el jugador pueda o no entender dicho mapa (Forest A., 2011).

III. METODOLOGÍA

La presente investigación adopta un enfoque cuantitativo experimental aplicado, centrado en el desarrollo de un algoritmo de PCG dentro del motor de videojuegos Unity. El objetivo principal es la implementación de una fórmula matemática unificada definida a trozos que permita instanciar mapas anillares de geometría modular, resolviendo la complejidad de posicionamiento mediante la segmentación del problema en procesos geométricos simples.

A. Materiales y Herramientas

Para el desarrollo del algoritmo y la validación de las fórmulas propuestas, se utilizaron los siguientes recursos de software y hardware:

Motor de desarrollo: Se utilizó Unity 3D, seleccionado por ser una de las herramientas más prevalentes en la industria y por su capacidad para manejar mecánicas esenciales y lógica matemática compleja mediante scripts.

Lenguaje de programación: C#, nativo de Unity, utilizado para traducir las ecuaciones matemáticas en instrucciones lógicas de instanciación.

Entorno de geometría modular: Se diseñaron "prefabs" (prefabricados) de baldosas geométricas (polígonos de *N* lados) para representar visualmente los cálculos.

B. Procedimiento matemático y algorítmico

El procedimiento central de esta metodología se basa en una función matemática que determina la posición vectorial *P* de cada módulo (baldosa) en el espacio 3D. Dado que las reglas de teselación varían según la figura geométrica, se diseñó una "Función definida a trozos" que altera su comportamiento según el número de lados *N* de la figura.

C. Definición de variables

Para todas las formulaciones subsiguientes, se definen los siguientes parámetros de entrada:

- *N*: Número de lados del polígono (e.g., 3, 4, 6).
- $R_{baldosa}$: Radio de la baldosa (distancia del centro al vértice).
- *g*: *Spacing* o brecha de separación entre baldosas.
- *R*: Índice del anillo actual (radio de la fila, donde $R \in Z^{+}$).
- *k*: Índice de la baldosa actual dentro del anillo *R*.

D. Geometría poligonal general ($N \geq 5$)

Para polígonos complejos como hexágonos, pentágonos, octágonos, el algoritmo utiliza coordenadas polares para recorrer el perímetro. Este método es esencial para estructuras que requieren una adaptación clara de navegación para NPCs y jugadores.

1. Cálculo de la zancada (*S*):

   Se determina la distancia de avance basándose en la apotema *A*.

$$A = r_{baldosa} \cdot \cos\left(\frac{\pi}{N}\right)$$

$$S = 2A + g$$

2. Determinación vectorial ($\vec{P}$): El algoritmo calcula los vectores de posición de dos vértices "ideales" ($\overrightarrow{V_{inicio}}$ y $\overrightarrow{V_{fin}}$) del anillo actual y aplica una interpolación lineal (Lerp) basada en el índice *k*.

Sean $\theta_{inicio}\ y\ \theta_{fin}$ los ángulos de los vértices que delimitan el segmento actual:

$$\overrightarrow{V_{inicio}} = R \cdot S \cdot \langle \cos(\theta_{inicio}), 0, \sin(\theta_{inicio}) \rangle$$

$$\overrightarrow{V_{fin}} = R \cdot S \cdot \langle \cos(\theta_{fin}), 0, \sin(\theta_{fin}) \rangle$$

La posición final se obtiene mediante:

$$\vec{P} = \overrightarrow{V_{inicio}} + \frac{k}{R}\left(\overrightarrow{V_{fin}} - \overrightarrow{V_{inicio}}\right)$$

E. Geometría cuadrangular ($N = 4$)

Para rejillas cuadradas, se optó por una disposición tipo "diamante" (rotación de 45°). Esto simplifica el cálculo a una rejilla cartesiana alineada a los ejes X y Z, eliminando la necesidad de cálculos trigonométricos complejos en cada iteración.

1. Cálculo de la zancada (*S*):

$$S = \left(r_{baldosa} \cdot \sqrt{2}\right) + g$$

2. Determinación vectorial ($\vec{P}$):

Se asignan coordenadas enteras *x* y *z* que oscilan entre -*R* y *R*.

$$\vec{P} = \langle x \cdot S,\ 0,\ z \cdot S \rangle$$

F. Geometría triangular ($\mathbf{N = 3}$)

La teselación triangular presentó la mayor complejidad lógica debido a la inversión de figuras para rellenar vacíos (teselación piramidal). El procedimiento distingue entre triángulos "Normales" y triángulos de "Relleno Invertido".

1. Dimensiones del paso:

Se calculan pasos diferenciados para los ejes horizontal (S_x) y vertical (S_z):

$$S_x = r_{baldosa} \cdot \sqrt{3} + g$$

$$S_z = 1.5 \cdot r_{baldosa} + (g \cdot \sin(60°))$$

2. Cálculo base:

Se establece la posición en el eje *x* partiendo del extremo izquierdo de la fila (X_{inicio}):

$$x = X_{inicio} + (k \cdot S_x)$$
$$z = -R \cdot S_z$$

3. Lógica condicional de instanciación:

El algoritmo evalúa si la baldosa actual *k* corresponde a un espacio de relleno.

- Si es Invertida (Relleno): Se aplica un desplazamiento (offset) de medio paso horizontal y medio radio vertical para encajar en el hueco superior.

$$\overrightarrow{P_{invertida}} = \left\langle x + \frac{S_x}{2},\ 0,\ z + \frac{r_{baldosa}}{2} \right\rangle$$

- Si es Normal:

$$\overrightarrow{P_{normal}} = \langle x,\ 0,\ z \rangle$$

IV. Resultados y discusion

A. *Geometría poligonal general ($N \geq 5$)*

Utilizando las fórmulas que calculaban el espaciado entre cada baldosa para la figura modular mediante el apotema de cada baldosa, se ha logrado diseñar los mapas modulares con PCG mediante anillos. Esta fórmula general permitió crear mapas hechas de cualquier figura de 5 o más lados. (Figura 1).

B. *Geometría cuadrangular ($N = 4$)*

Mediante las fórmulas presentadas anteriormente, que aprovechan los ángulos rectos de estas figuras para calibrar de manera correcta el posicionamiento de las baldosas, se logró crear el mapa de manera cuadrada con geometría modular. (Figura 2).

C. *Geometría triangular ($N = 3$)*

El uso de la teselación triangular en las fórmulas, que permite al programa identificar cuales triángulos tienen que ser invertidos, permitió obtener el mapa de forma triangular en diseño modular, obteniendo la colocación exacta de los triángulos, tantos invertidos como normales. (Figura 3).

D. *Conclusiones*

Con el programa creado, se logró implementar el código de PCG para los mapas modulares de manera eficaz, validando el uso de geometría modular para la creación de mapas anillares en videojuegos de estrategia, tanto por turnos como en tiempo real, en 3D. Este diseño modular permitió que el mapa evolucione algorítmicamente, resolviendo la dificultad histórica de crear mapas donde el jugador pueda orientarse fácilmente.

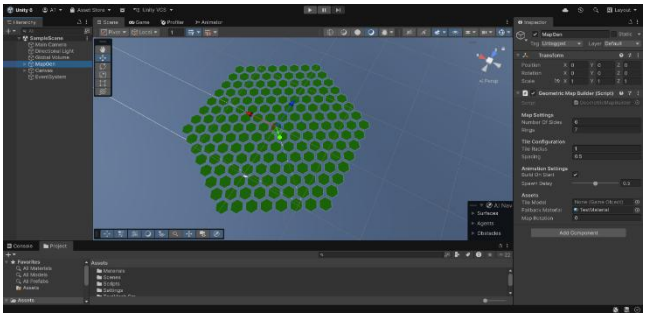
Al utilizar fórmulas matemáticas directas en lugar de grandes matrices de datos pre-guardados, se pudo obtener la agilidad de la producción, evitando ineficiencias de código comunes en proyectos de Unity. La segmentación del código según la categoría de polígonos con las cuales está trabajando el programa también ayudó con rendimiento, previniendo un uso excesivo de procesamiento para forzar que una sola formula funcione para todas las figuras poligonales. Las fórmulas de los triángulos lograron cortar procesos de revisión de parte del código para verificar cuales triángulos tenían que estar reposicionados; además, la formula general para figuras de 5 y 7 lados en adelante simplificaron el código para ayudarnos a crear una mayor cantidad de mapas con mayor número de lados para las figuras. Finalmente, las fórmulas de los cuadrados aprovecharon la estructura de estas figuras para simplificar el proceso de elaboración del mapa, mediante cálculo de vectores en vez de utilizar la apotema de la figura.

La implementación de una configuración de espaciado entre cada baldosa permitió variar el estilo del mapa creado, que también ayudaría a implementar los mapas en diferentes partes del juego, como un árbol de habilidades (sin espaciado), o una introducción a una arena (mucho espaciado). La habilidad de poder seleccionar cada celda de la cual está conformada el mapa da el beneficio adicional de agilizar mecánicas de movimiento de unidades a través de cualquier parte del mapa, algo esencial en los juegos de estrategia por turnos.

Todos estos detalles permitirían a más desarrolladores crear juegos en 3D mediante Unity que requieren este estilo de mapas, especialmente juegos de estrategia. Esto cumple el objetivo de ampliar la cantidad de personas quienes puedes crear su propio videojuego, acortando los tiempos de desarrollo y reduciendo los costos de producción.

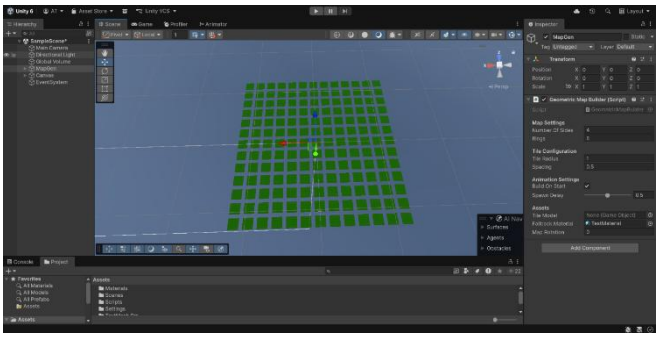
V. ANEXOS

A. *Figura 1*



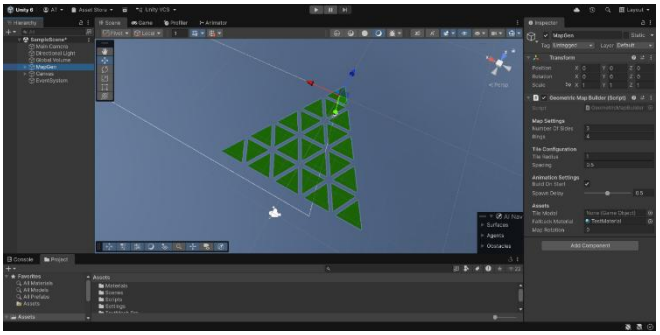
Nota: Muestra de generación de un mapa hexagonal con 7 anillos

B. *Figura 2*



Nota: Muestra de un mapa en forma de cuadrado con 6 anillos

C. *Figura 3*



Nota: Muestra de un mapa en forma de triángulo con 4 anillos; los triángulos se invierten de acuerdo con el posicionamiento de las baldosas.

VI. Referencias

[1] Borrelli, A., Nardone, V., Di Lucca, G. A., Canfora, G., & Di Penta, M. (2020). Detecting Video Game-Specific Bad Smells in Unity Projects. *Proceedings - 2020 IEEE/ACM 17th International Conference on Mining Software Repositories, MSR 2020*, 198–208. https://doi.org/10.1145/3379597.3387454

[2] Fabricatore, C. (2007). GAMEPLAY AND GAME MECHANICS DESIGN GAMEPLAY AND GAME MECHANICS DESIGN: A KEY TO QUALITY IN VIDEOGAMES. https://doi.org/doi.org/10.13140/RG.2.1.1125.4167

[3] Forest A. (2011). Which Way From Here? Which Way From Here?

[4] Foxman, M. (2019). United We Stand: Platforms, Tools and Innovation With the Unity Game Engine. *Social Media and Society*, *5*(4). https://doi.org/10.1177/2056305119880177

[5] Gordo, G. B. (2020). *Trabajo Fin de Grado VIDEOJUEGO de PLATAFORMAS 2D en UNITY*.

[6] Haas, J. (2014). A History of the Unity Game Engine An Interactive Qualifying Project Submitted to the Faculty of WORCESTER POLYTECHNIC INSTITUTE in partial fulfillment of the requirements for graduation.

[7] Hartsook, K., Zook, A., Das, S., & Riedl, M. O. (2011). Toward supporting stories with procedurally generated game worlds. *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, 297–304. https://doi.org/10.1109/CIG.2011.6032020

[8] Kaźmierczak, R., Skowroński, R., Kowalczyk, C., & Grunwald, G. (2024). Creating Interactive Scenes in 3D Educational Games: Using Narrative and Technology to Explore History and Culture. *Applied Sciences (Switzerland)*, *14*(11). https://doi.org/10.3390/app14114795

[9] Kelly G., M. H. (2006). *Procedural_City_Generation_Survey*. https://doi.org/10.21427/D76M9P

[10] Khan, N., & Rahman, A. U. (2018). Rethinking the Mini-Map: A Navigational Aid to Support Spatial Learning in Urban Game Environments. *International Journal of Human-Computer Interaction*, *34*(12), 1135–1147. https://doi.org/10.1080/10447318.2017.1418804

[11] Kim, J. H., & Sung, H. (2024). A Hexagon Sensor and A Layer-Based Conversion Method for Hexagon Clusters. *Information (Switzerland)*, *15*(12). https://doi.org/10.3390/info15120747

[12] Lai, C. H., & Hu, P. Y. (2025). The Gaming Revolution in History Education: The Practice and Challenges of Integrating Game-Based Learning into Formal Education. In *Information (Switzerland)* (Vol. 16, Issue 6). Multidisciplinary Digital Publishing Institute (MDPI). https://doi.org/10.3390/info16060490

[13] Lammes, S., & Wilmott, C. (2018). The map as playground: Location-based games as cartographical practices. *Convergence: The International Journal of Research into New Media Technologies*, *24*(6), 648–665. https://doi.org/10.1177/1354856516679596

[14] Mohammed Abdul, S. S., Shrestha, A., & Yong, J. (2025). Toward the Mass Adoption of Blockchain: Cross-Industry Insights from DeFi, Gaming, and Data Analytics. In *Big Data and Cognitive Computing* (Vol. 9, Issue 7). Multidisciplinary Digital Publishing Institute (MDPI). https://doi.org/10.3390/bdcc9070178

[15] Mukherjee, S. (2015). The playing fields of Empire: Empire and spatiality in video games. *Journal of Gaming and Virtual Worlds*, *7*(3), 299–315. https://doi.org/10.1386/jgvw.7.3.299_1

[16] Politowski, C., Petrillo, F., Montandon, J. E., Valente, M. T., & Guéhéneuc, Y.-G. (2020). *Are Game Engines Software Frameworks? A Three-perspective Study*. https://doi.org/10.1016/j.jss.2020.110846

[17] Rabin S. (2010). Game Programming Gems 1.

[18] Sahr, K. (2011). HEXAGONAL DISCRETE GLOBAL GRID SYSTEMS FOR GEOSPATIAL COMPUTING. In *Cartography and Remote Sensing* (Vol. 22).

[19] Schell, J. (2008). The Art of Game Design: A Book of Lenses.

[20] Sharma, A., Dalmia, A., Kazemi, M., Zouaq, A., & Pal, C. J. (2024). *GeoCoder: Solving Geometry Problems by Generating Modular Code through Vision-Language Models*. http://arxiv.org/abs/2410.13510

[21] Shubin, A. V., & Kugurakova, V. V. (2024). Designing a Tool for Creating Gameplay through the Systematization of Game Mechanics. *Automatic Documentation and Mathematical Linguistics*, *58*(S5), S299–S306. https://doi.org/10.3103/s0005105525700384

[22] Smelik, R. M., Jan de Kraker, K., Groenewegen, S. A., Tutenel, T., & Bidarra, R. (2009). *A Survey of Procedural Methods for Terrain Modelling* *.

[23] Van Der Linden, R., Lopes, R., & Bidarra, R. (2014). Procedural generation of dungeons. In *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAMES* (Vol. 6, Number 1).

[24] Walfisz, M., Zackariasson, P., & Wilson, T. L. (2006). Real-time strategy: Evolutionary game development. *Business Horizons*, *49*(6), 487–498. https://doi.org/10.1016/j.bushor.2006.04.001

[25] Wang, Y., Luo, J., Gaier, A., Atherton, E., & Koch, H. (2024). *PlotMap: Automated Layout Design for Building Game Worlds*. http://arxiv.org/abs/2309.15242

[26] Yallico, A. A., Chávez, J. I., & Barrientos, A. (2025). Development of a real-time strategy genre video game for PC using Unity and Blender. *Memorias de La Conferencia Iberoamericana de Complejidad, Informatica y Cibernetica, CICIC*, *2025-March*, 167–173. https://doi.org/10.54808/CICIC2025.01.167

[27] Zubek, Robert. (2020). *Elements of game design*. The MIT Press.