Primer acercamiento a la generación procedural de contenido, enfocado en mapas de altura con el algoritmo Diamante Cuadrado.

First approach to procedural content generation, focusing on height maps with the Diamond Square algorithm.

Juan Aguilera Huerta[1]
[1]Licenciatura en Ingeniería en Sistemas Computacionales
División de Ingenierías, Campus Irapuato-Salamanca
Universidad de Guanajuato

j.aguilerahuerta@ugto.mx[1]

Resumen

La generación de contenido procedural, aunque no está definida de manera formal, se enfoca en crear contenido a través de procedimientos mediante algoritmos que siguen ciertas restricciones. Es un área de estudio que se compone de muchas otras como las estructuras de datos, algoritmos, geometría computacional, computación estética y un gran trabajo de diseño previo. El contenido generado a través de estos métodos aún no está muy bien posicionado en la industria, sigue siendo delegado a tareas más sencillas y de menor riesgo para que no perjudique la calidad del producto que se puede obtener mediante procesos manuales mejor controlados.

Para seguir mejorando estos procesos, comunicar e invitar a otras personas a este campo, se decidió explorar un algoritmo introductorio en la generación de mapas de altura, el algoritmo Diamante Cuadrado que ya tiene más de 30 años desde que Fournier, Fussell y Carpenter en 1982 lo presentaron. A través de ello, se han diseñado diferentes versiones que se basan en reglas y limitaciones a cambio de obtener resultados menos generales, pero de mayor calidad.

Unity contribuye a la democratización del conocimiento y herramientas para el desarrollo de aplicaciones inmersivas e interactivas. El objetivo de la investigación se enfoca en aprovechar estos medios, analizar y comprender el alcance y limitaciones del algoritmo para consecuentemente, generar contenido digital que de manera ideal pueda ser eficiente en cuestiones de tiempo, producción, diseño y adaptabilidad según las necesidades del usuario para contribuir a la creación de un espacio inmersivo.

Palabras clave: algoritmo; mapa de alturas; lúdico; multimediático; procedural; gameplay; dinámicas; pseudoaleatorio;

Introducción

A pesar de que no exista una definición formal, la generación procedural de contenido se define como una forma de creación automática de contenido mediante el uso de algoritmos (o procedimientos) en lugar de crearlo de forma manual. Este contenido (texturas, terrenos, narrativas, plantas, ecosistemas, ríos, redes de caminos, mecánicas, dinámicas, etc.) es generado con entrada de información nula, limitada o indirecta de los programadores. Sus principales fortalezas recaen en sus capacidades para comprimir información (Sánchez et al. 2013).

Es un ámbito tecnológico cada vez más importante en el diseño moderno de la interacción persona-computadora. La personalización de la experiencia del usuario mediante el modelado afectivo y cognitivo, junto con el ajuste en tiempo real de los contenidos en función de las necesidades y preferencias del usuario son pasos importantes hacia una generación procedural de contenido (PCG) eficaz y significativa (G. N. Yannakakis and J. Togelius, 2011).

El videojuego es uno de los medios más representativos cuando se trata de aplicaciones de la creación de contenido. Así mismo, uno de sus puntos más fuertes es la síntesis de emociones complejas a través de patrones afectivos y cognitivos que se generan durante el gameplay del mismo. Para lograr este objetivo, hay muchos elementos que deben coexistir en una sinergia muy precisa. Dentro de esos elementos los terrenos y todo el ambiente en el que se desarrolla la historia o idea de un videojuego juegan un papel crucial.

La generación procedural ha evolucionado significativamente a lo largo de los años y hay numerosos ejemplos aplicados donde se ha generado contenido para diferentes videojuegos como Rogue (1980), Diablo (1996), Civilization (1991), Minecraft (2011), Hades (2020). Cada título ha enfocado y aprovechado la PCG según sus necesidades, en el caso de *Rogue (figura 1),* genera dungueons con ciertas características que hacen atractivo el gameplay para el jugador, enemigos, objetos, potenciadores y otros elementos que conducen al jugador a tomar decisiones que afectan directamente el curso de la partida y obligan al jugador a prestar atención a lo que ocurre en el mundo del juego.

Figura 1.Captura de un nivel en Rogue.

El contenido de los juegos se refiere a todos los aspectos de un juego que afectan al gameplay pero estrictamente no son procesos determinísticos como pueden ser el terreno, mapas, niveles, historias, diálogos, misiones, jugadores, dinámicas, música, armas, iluminación y otros componentes. Los esfuerzos y tiempos que se requieren para la creación de este tipo de contenido representan una parte muy extensa del ciclo de desarrollo de un videojuego. A pesar de contar con herramientas gráficas más avanzadas que facilitan muchas partes del trabajo, automatizar estos procesos sigue siendo una tarea muy compleja dentro de la industria.

Diseñar estas experiencias inmersivas ahora es mucho más viable gracias a la inversión que genera la industria de los videojuegos en el sector tecnológico para crear hardware más potente y expandir los límites. Cada vez es más viable que una persona que se dedique al diseño de juegos pueda experimentar con ideas y conceptos más complejos que se adapten a habilidades, preferencias y emociones de diferentes grupos de jugadores.

Los motores de videojuegos o motores gráficos nos permiten visualizar y crear de manera digital esta clase de contenidos para videojuegos, combinarlos y crear mundos libres de limitaciones en la mayoría de los casos. Aunque cada motor busca cubrir la mayor cantidad de áreas para la producción de estas experiencias inmersivas, hay varios puntos fuertes que se pueden destacar de cada uno:

- Unreal Engine. Gráficos de alta calidad y juegos en tres dimensiones.
- Unity. Juegos en dos dimensiones y experiencia de usuario amigable.
- Godot: Código abierto para juegos en dos y tres dimensiones.
- Twine: Código abierto para ficciones interactivas e historias no lineales.

Figura 2. Ejemplos de aplicaciones en videojuegos procedurales (Shadows of Doubt y Panoramical).

Obtener un buen resultado en estos procesos siempre va ligado con restricciones claras del producto que se busca, para los terrenos se pueden tomar en cuenta aspectos como la compresión de la distancia horizontal, aumento de las pendientes, clima idealizado, vegetación y características hídricas simplificadas, subpoblación y segregación espacial de las minorías étnicas. Las necesidades y características que cubre un terreno se pueden enlistar de la siguiente manera (Fraile-Jurado, P., 2023):

- Tamaño del mapa.
- Velocidad de tránsito.
- Realismo del terreno.
- Verosimilitud del clima, agua, vegetación.
- Realismo de densidad de población, distribución y uso de terrenos.

En base a las diferentes aplicaciones que se les ha dado dentro de la industria, sabemos que en los videojuegos se crean diferentes interpretaciones de la percepción que se tiene sobre la realidad, aportando incluso diferentes perspectivas hacia este ciclo de persona-ambiente que genera mayor inmersión.

En esta investigación se explorará la aplicación del algoritmo Diamante Cuadrado (Fournier et al., 1980) para la generación de terrenos en tres dimensiones, un proceso controlado por restricciones más básicos e introductorias como el tamaño de este, rangos de altura e identificación de materiales simulen algunos de la vida real bajo ciertos contextos.

Metodología

Buscando hacer este proceso lo más accesible y amigable para la mayoría de las personas, como se mencionó anteriormente, se optó por usar el motor de Unity y el lenguaje de C# para la implementación de este algoritmo. Aunque todo el funcionamiento del código se presentará a continuación en pseudo código, pueden encontrar el vínculo al repositorio del proyecto realizado en Unity aquí enlace.

Podemos dividir el proceso para generar el terreno en tres etapas.

1. **Implementación del algoritmo.** Consiste en crear y manipular la matriz de *n* dimensiones para generar un terreno con escarpaduras lo menos abruptas posibles.
2. **Construcción del terreno.** Instanciamos diversos objetos en Unity para representar la matriz generada como resultado de aplicarle el Diamante Cuadrado.
3. **Ajustes de parámetros para controlar la generación del terreno.**

1. Implementación del algoritmo

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:

1.1 Paso del diamante

El 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.

1.2 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.

En la Figura 2 se presentan 2 iteraciones del algoritmo. Una primera iteración para el paso Diamante (2.a) donde se definen en las fichas negras las esquinas que delimitan el tamaño de nuestro terreno con valores pseudo aleatorios con los que calcularemos todos los demás valores de nuestra matriz en los otros pasos. En la ficha naranja se representa el promedio de esas cuatro esquinas al que se le agrega un valor aleatorio definido libremente por el usuario. En la segunda iteración para el paso Cuadrado (2.b) ahora se almacenan los promedios en las fichas azules de la misma manera, repetimos el mismo proceso en el paso (2.c) y así secuencialmente hasta llenar todos los espacios de la matriz.

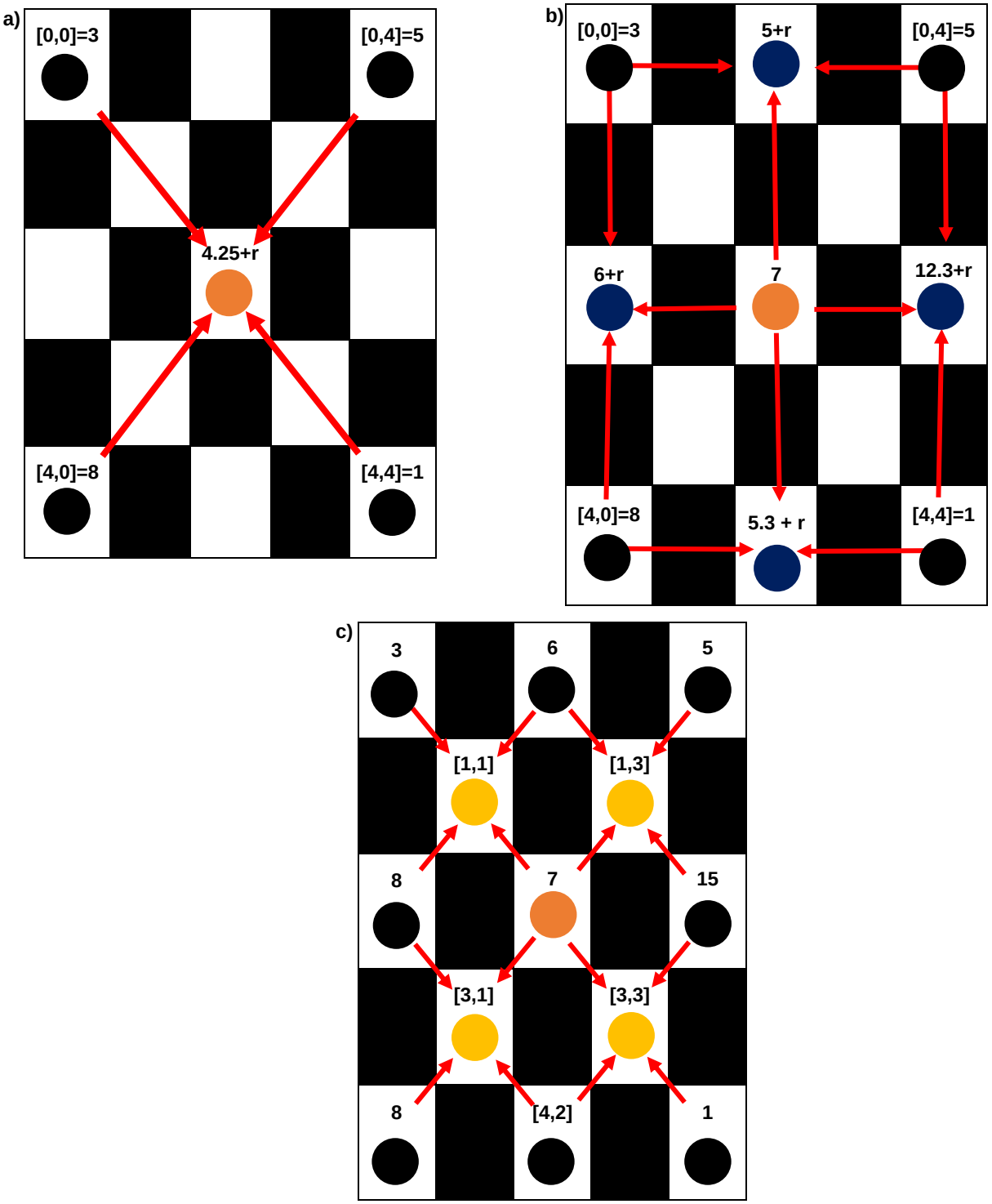

Figura 4. a) Diamond Step, b) Square Step, c) Diamond Step

Esto se traduce en el siguiente pseudocódigo:

```
Función DiamondSquareStep() -> Matriz de flotantes
    Definir m_half como m_chunkSize / 2

    // Paso Diamond
    Para y desde m_half hasta m_heightMapSize - 1 con incremento m_chunkSize
        Para x desde m_half hasta m_heightMapSize - 1 con incremento m_chunkSize
            // Calculando el promedio de los puntos del diamante
            Definir avg como (heightMap[x - m_half, y - m_half] +
                              heightMap[x + m_half, y - m_half] +
                              heightMap[x - m_half, y + m_half] +
                              heightMap[x + m_half, y + m_half]) / 4.0
            Definir randomAdd como Random.Range(-m_randomRange, m_randomRange)
            m_heightMap[x, y] <- avg + randomAdd
    Fin Para

       // Paso Square
    Para y desde 0 hasta m_heightMapSize - 1 con incremento m_half
        Para x desde (y + m_half) % m_chunkSize hasta m_heightMapSize - 1 con incremento
m_chunkSize
            // Calculando el promedio de las esquinas
            Definir avg como (heightMap[(x - m_half + m_heightMapSize) % m_heightMapSize,
y] +
                              heightMap[(x + m_half) % m_heightMapSize, y] +
                              heightMap[x, (y + m_half) % m_heightMapSize] +
                              heightMap[x,   (y   -   m_half   +   m_heightMapSize)   %
m_heightMapSize]) / 4.0
            Definir randomAdd como Random.Range(-m_randomRange, m_randomRange)
            heightMap[x, y] <- avg + randomAdd
    Fin Para

    Devolver heightMap
Fin DiamondSquare
```

Es importante que antes de todo esto, se hayan asignado los valores para las cuatro esquinas que delimitan nuestro terreno

```
Procedimiento SetRandomCorners(Matriz de flotantes heightMap, flotante randomRange,
entero heightMapSize)
    // Valores aleatorios en las esquinas del cuadrado
    heightMap[0, 0] <- Random.Range(-randomRange, randomRange)
    heightMap[0, heightMapSize - 1] <- Random.Range(-randomRange, randomRange)
    heightMap[heightMapSize - 1, 0] <- Random.Range(-randomRange, randomRange)
    heightMap[heightMapSize  -  1,  heightMapSize  -  1]  <-  Random.Range(-randomRange,
randomRange)
Fin Procedimiento
```

2. 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 trozos, 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.

Tabla 1. Rangos para la función continua a trozos de la altura con relación al tamaño del terreno.

Tamaño del terreno ($2^n + 1$)	Rango aleatorio
3	2
5	4
9	10
17	15
33	20
65	30
129	35
257	40
513	50

3. 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 objeto 3D 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.

```
Procedimiento BuildHeightmap(Matriz de flotantes heightMap, Prefab de GameObject cubePrefab)

    // Variables para filas y columnas
    Entero heightmap_rows <- ObtenerLongitud(heightMap, 0)
    Entero heightmap_columns <- ObtenerLongitud(heightMap, 1)

    // Recorremos toda la matriz del mapa de altura para cada fila desde 0 hasta heightmap_rows - 1
        Para cada columna desde 0 hasta heightmap_columns - 1
            // Asignamos la posición donde vamos a instanciar el prefab
            Vector3 position <- Nuevo Vector3(columna * 1, 10.0, fila * 1)
            GameObject cube <- Instanciar(cubePrefab, posición, Identidad)
            // Esta función es para evitar tener cubos con altura de 0, lo que provoca un bug visual que muestra el cubo de color negro
            ArreglarTamañoDelCubo(heightMap, fila, columna, cubo)
            // Para localizarlos más fácil, el nombre de cada cubo es su altura correspondiente
            AsignarNombre(cubo, ConvertirATexto(heightMap[fila, columna]))

            // Función para asignar el material al cubo según su altura, representando "agua", "tierra" o "pasto"
            AsignarMaterial(heightMap, fila, columna, cubo)
        Fin Para
    Fin Para
Fin Procedimiento
```

El resto de las partes del programa son para configurar la interfaz de usuario, pero para los fines de esta investigación no se consideró necesario incluir esa partes, recordando que cualquier duda puede ser consultada directamente en el repositorio con todo el código del proyecto.

Resultados

Estos son algunos resultados de terrenos generados en tiempo real con algunas configuraciones específicas, podemos observar que los terrenos se generan con alturas coherentes sin cambios abruptos.

Las siguientes figuras (3, 4 y 5) representan algunos ejemplos de las pruebas que se hicieron con terrenos antes y después de aplicarles algún material para simular el efecto de tierra, agua y pasto.

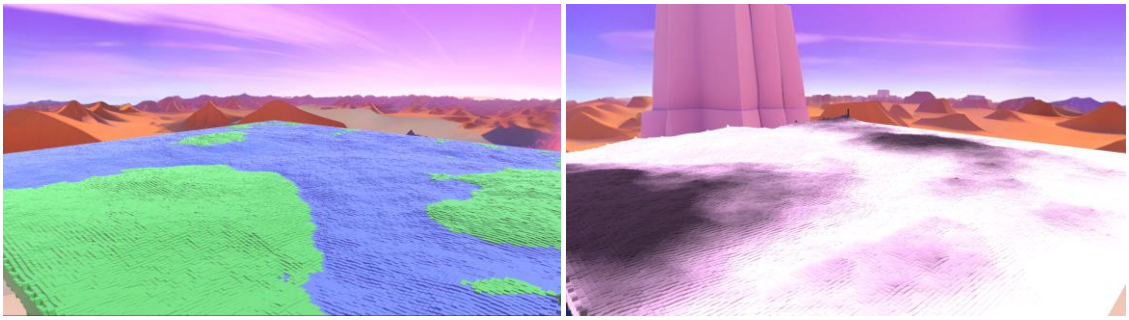

Figura5. Generación de terreno (con materiales y sin materiales) de 257x257 y un rango aleatorio de -22.45 a 22.45.

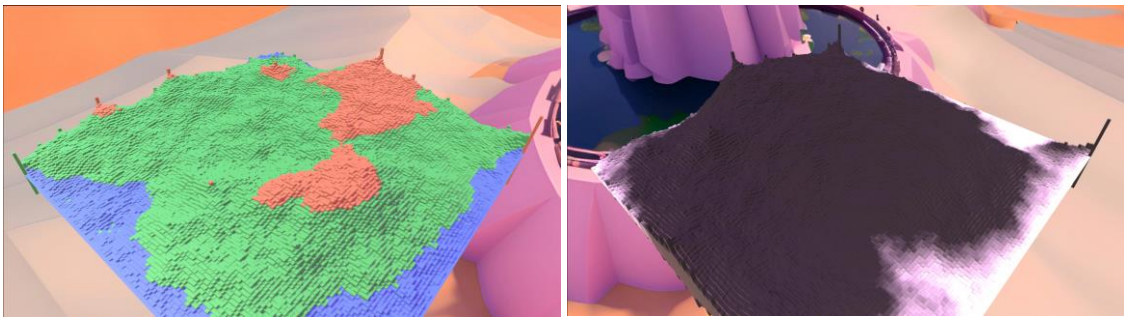

Figura 6. Generación de terreno (con materiales y sin materiales) de 129x129 y un rango aleatorio de -35 a 35.

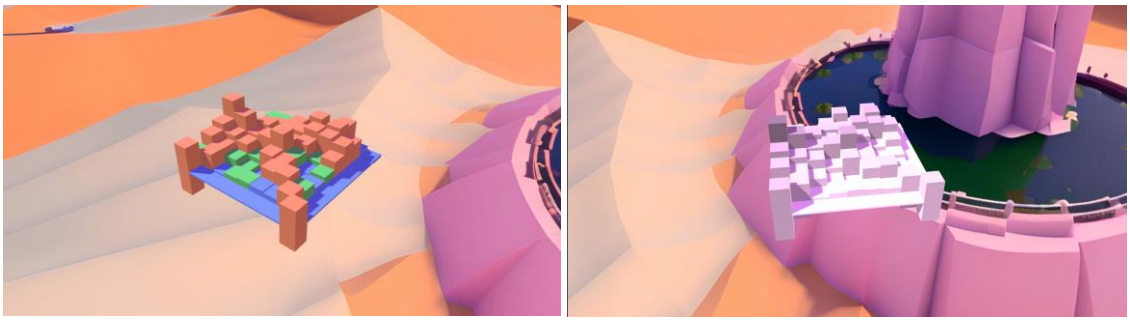

Figura 7. Generación de terreno (con materiales y sin materiales) de 9x9 con un rango aleatorio de -4.75 a 4.75.

Conclusiones

La programación de videojuegos es un medio didáctico excelente para introducir a las personas en estas áreas de la computación y la generación procedural de contenido es una aproximación muy buena para combinar lo mejor de ambos campos. Videojuegos y algoritmia.

En cuanto a limitaciones, se puede abordar el tema de los conocimientos técnicos para optimizar el código, para investigar más a fondo sobre la triangulación de Delaunay para obtener muchos mejores resultados y no dejar que todo recaiga en lo pseudo aleatorio del proceso. Compartir este trabajo y lograr que más personas se adentren a estos temas tan interesantes.

La exploración de estos temas han sido muy gratificantes durante la investigación, el panorama que hay detrás de todos esos procesos que son fundamentales en industrias creativas y artísticas digitales para reducir tiempos y costos de producción, pero no solo eso, que nos permiten una mayor accesibilidad en la creación de contenido, que fomenta la libertad creativa de cualquier persona para crear experiencias más arriesgadas sobre aquellas que las grandes empresas buscan para generar ganancias en sus inversiones.

Al tratarse de uno de los algoritmos básicos cuando se habla de generación procedural de contenido, el producto desarrollado sin duda es una herramienta muy útil para que las personas que la usen puedan generar contenido instantáneamente según sus necesidades, así como darse cuenta de que, siguiendo ciertos procesos, se puede manipular una simple matriz de dos dimensiones y generar lugares que pueden ser habitables por más objetos. Las bases de la programación juegan un papel muy importante, pero es gracias a herramientas tan didácticas como los motores de videojuegos, que tú tienes la posibilidad de crear y sintetizar comportamientos o procesos con algoritmos, no se trata de minimizar todo a ciertas condicionales, sino de analizar de verdad, de qué se componen las cosas y cómo pueden ser replicadas a través de bloques de código.
A pesar de la limitación tan grande que se tuvo para implementar el algoritmo en un campo donde no tenía nada de experiencia, se obtuvo un prototipo sobre el que se pueden seguir implementando mejores, como pudiera ser la triangulación de Delaunay para formar superficies mucho más suaves y naturales (Short, T., & Adams, T., 2017)

La exploración y perfeccionamiento que hacen a la generación procedural de contenido aún inestable ante ciertas situaciones es algo que se sigue investigando y desarrollando, este proyecto introduce al lector a un mundo lleno de algoritmos, principios de diseño, posibilidades de comprimir objetos en procesos que repliquen y generen nuevos resultados. Las posibilidades a partir de aquí se aproximan a la dedicación que tengan las personas lectoras por buscar algoritmos más complejos y desafiantes.

Bibliografía/Referencias

Archer, T. (2011, April). Procedurally generating terrain. In 44th annual midwest instruction and computing symposium, Duluth (pp. 378-393).

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.

Gomes, A., & Mendes, A. J. (2007, September). Learning to program-difficulties and solutions. In International Conference on Engineering Education–ICEE (Vol. 7).

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 9(1), 1-22.

Johns, J. (2006). Video games production networks: value capture, power relations and embeddedness. Journal of economic geography, 6(2), 151-180.

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

Short, T., & Adams, T. (Eds.). (2017). Procedural generation in game design. CRC Press.

Soloway, E. (1986). Learning to program= learning to construct mechanisms and explanations. Communications of the ACM, 29(9), 850-858.

Tisza, G., & Markopoulos, P. (2021). Understanding the role of fun in learning to code. International Journal of Child-Computer Interaction, 28, 100270.