Departamento de Ingeniería Informática
y Ciencias de la Computación
Universidad de Concepción

"Estudio de IA generativa para distribución y mapeado en videojuegos"

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Informático.

Autor: **Víctor Mora Lara**

Profesor Guía: **Dr. Julio Godoy Del Campo**

Abril 2026

Concepción Chile.

Índice

1. Introducción

En la sociedad actual los videojuegos son utilizados cotidianamente por diversos grupos de personas y con diversos propósitos, ya sean educativos, recreativos u otros. Estas aplicaciones se han instaurado de forma persistente e ininterrumpida, además, tienen una doble naturaleza lúdico-técnica y son una de las más claras manifestaciones de esta nueva sociedad, caracterizada por la democratización del acceso a la información y las producciones culturales (Moreno y Venegas, 2020). Lo anterior, junto al avance en las tecnologías y la propia exploración artística y técnica de los desarrolladores de videojuegos, ha llevado a la categorización de videojuegos según el área o temática en la cual se enfocan. Así, es como se adoptan diversos métodos y conveniencias comunes en el desarrollo de aquellos, tales como la disposición del espacio jugable, los tipos de mecánicas de interacción, los sistemas de progresión y los mapeados, entre otros elementos.

El mapa y entorno de los niveles de un videojuego son muy importantes para ofrecer una experiencia completa, al ofrecer al jugador espacios navegables en los que progresar dentro de estos (Servet, 2014). De acuerdo a esto, gran parte del esfuerzo de desarrollar un videojuego se vierte en el mapeado, siendo este el entorno o espacios jugables donde se desarrolla la experiencia del jugador, compuestos por la disposición de escenarios, habitaciones, caminos y puntos de interés que forman parte de un nivel, de forma que el jugador se desplace, explore e interactúe con el mundo virtual en base a estos factores.

Es así, que se presenta un impacto directo del entorno en la jugabilidad y experiencia del usuario, siendo la continuidad de los niveles un aspecto importante para un progreso coherente e inmersivo (Servet, 2014). Es debido a esta importancia, que se presentan varios puntos de interés al desarrollar un mapeado. De esta manera, el área algorítmica del diseño de mapeados será el enfoque de esta investigación.

Dentro del amplio contexto en el cual se desarrollan los videojuegos, la generación procedural de contenido es una técnica ampliamente utilizada en el desarrollo de

videojuegos para producir contenido de manera automática, ofreciendo ventajas como la rejugabilidad (Bea, 2017). Este enfoque permite reducir el tiempo de diseño manual, haciendo uso de reglas algorítmicas que garantizan variabilidad en diferentes ejecuciones. Sin embargo, los resultados están estrictamente atados a los parámetros definidos por el diseñador, pudiendo limitar la originalidad y diversidad estructural en función de los límites entregados al algoritmo.

En consideración de lo planteado, la generación procedural se trata de un método que utiliza algoritmos recursivos y pseudoaleatorios para producir entornos dinámicos automáticamente, algo deseado especialmente en géneros que dependen de la rejugabilidad como los "roguelike" o de exploración libre (Cuadra, 2023). De esta manera, la generación procedural se instala como uno de los métodos ideales para el diseño de ciertos estilos de juegos en el presente.

Además, se ha demostrado que la generación procedural puede crear mundos virtuales vastos y consistentes, en ocasiones simulando un universo y haciendo uso de una gran combinación de posibilidades. Esto se puede observar en títulos comerciales como *No Man's Sky*, el cual hace uso de Perlin Noise, un algoritmo de ruido pseudoaleatorio que entrega patrones visualmente naturales que, junto a diversas fórmulas matemáticas, logran simular un universo de 18 quintillones de planetas posibles (Shen, 2022). O también, *Spelunky,* que presenta un componente algorítmico más sencillo basado en random walker para su mapeado (Kazemi, 2013). Si bien este último es más simple, continúa ofreciendo amplias posibilidades. No obstante, esta cantidad de posibilidades conlleva la necesidad de establecer reglas y condiciones manuales, restringiendo la creatividad del sistema a un entorno de posibilidades conocido con el fin de evitar situaciones de poca coherencia o que podrían ocasionar errores dentro del videojuego, intercambiando variedad por estabilidad.

En paralelo, los modelos de lenguaje de gran escala (LLMs) se presentan como modelos de lenguajes estadísticos, los que codifican probabilidades de distribución sobre conjuntos de

palabras para simular un lenguaje (Douglas, 2023). Lo anterior, junto a técnicas de fine-tuning, que a su vez consisten en obtener un modelo especializado en una tarea acotada mediante el entrenamiento de un set de datos pequeño en un modelo de conocimiento generalizado (Church et al, 2021), han permitido a la IA reconocer patrones y generar resultados complejos con mayor coherencia contextual. Su capacidad de generalización abre la posibilidad de delegar parte del diseño de niveles a sistemas generativos basados en aprendizaje en lugar de reglas fijas.

2. Presentación del problema

Los avances recientes en inteligencia artificial generativa, particularmente en los modelos de lenguaje de gran escala (LLMs), han abierto nuevas posibilidades en la generación contextual de contenido. Como plantean Farrokhi y Zhao (2024), el uso de modelos de aprendizaje automático (LLM) para tareas generativas es un desarrollo reciente, con métodos relacionados propuestos alrededor del año 2015, más así desconocidos para el público general hasta hace unos cinco años, lo que actualmente está captando gran atención entre los investigadores en el área de generación de contenido procedural para juegos.

Para muchos videojuegos, especialmente aquellos que presentan un alto grado de aleatoriedad en la distribución de sus escenarios y por tanto, variedad en el diseño estos, es que se puede notar una exigencia importante en relación al tiempo, recursos utilizados y altísimos costos computacionales asociados. Como solución a esta complejidad creciente, se recurre a la generación de contenido procedural (Zhang et al, 2022). Sin embargo, dada la creciente complejidad de los videojuegos modernos, especialmente en la generación procedural dinámica de escenarios y elementos, es que surge la necesidad de explorar nuevas alternativas que amplíen las capacidades de los métodos actuales.

En este contexto, se plantea evaluar el desempeño generativo de los modelos de lenguaje de gran escala como posible complemento o alternativa de los algoritmos procedurales tradicionales, los cuales, pese a su eficacia, pueden presentar limitaciones en cuanto a diversidad estructural, coherencia narrativa y creatividad espacial según los videojuegos crecen y se modernizan (Pereira de Araujo, 2017). Lo anterior, junto a la necesidad de aplicar restricciones algorítmicas necesarias a los algoritmos procedurales, pueden generar patrones repetitivos perceptibles para el jugador cuando no se dispone de una lógica o configuración adecuadas, o la envergadura del videojuego hace evidentes patrones lógicos del algoritmo.

Este trabajo tiene como propósito principal determinar si la inteligencia artificial generativa puede ofrecer una alternativa viable o complementaria al enfoque procedural tradicional en la creación de mapas de videojuegos. Para llevar a cabo esta investigación, se adopta un enfoque cuantitativo y experimental, basado en la medición comparativa de métricas de desempeño. Al tratarse de una tecnología en crecimiento y con potencial, se hará uso de un LLM para comparar su desempeño contra algoritmos procedurales.

3. Objetivos del Proyecto

- Objetivo General

  Evaluar la capacidad de un modelo de inteligencia artificial generativa ajustado mediante fine-tuning para generar mapas 2D coherentes y compararlos con un algoritmo procedural de tipo Random Walker.

- Objetivos Específicos
  - Analizar los principales parámetros estructurales para la generación procedural de mapas 2D.
  - Implementar un modelo generativo (LLM) entrenado para la tarea de mapeado estructurado.
  - Desarrollar un entorno experimental que permita comparar ambos enfoques bajo métricas comunes.
  - Analizar resultados cuantitativos y cualitativos, considerando eficiencia, diversidad y coherencia estructural.

4. Estado del arte

Dado los temas de interés de esta investigación, se presenta la generación de contenido procedural y los modelos de lenguaje de gran escala (LLM). La generación de contenido procedural se comprende como una metodología ampliamente utilizada en la automatización de la creación de entornos, niveles y elementos de los videojuegos mediante algoritmos pseudo aleatorios y deterministas. Mientras que los LLM, debido a su reciente surgimiento, traen consigo nuevas posibilidades en la generación de contenido contextual y coherente.

Para motivos de la investigación, se realizará una indagación acerca de ciertos modelos de lenguaje y métodos de generación procedural existentes y conocidos en la actualidad. A partir de esto, es posible desarrollar una evaluación y selección de los modelos y métodos adecuados para los propósitos establecidos.

4.1 Modelos de lenguaje

Los LLM han presentado un gran impacto en la actualidad en consideración de los beneficios que otorgan. Se establece que han demostrado ser muy prometedores como asistentes de IA, siendo altamente capaces en tareas de razonamiento complejas que requieren conocimientos especializados en una amplia gama de campos, incluyendo dominios especializados como la programación (Touvron, et al. 2023). Por este motivo, los modelos de lenguaje son un complemento para el desarrollo de videojuegos.

En esta sección se indaga acerca de cinco modelos de lenguaje, para evaluar su pertinencia dentro de la investigación:

1. LLaMA 2: En primer lugar se presenta LLaMA 2, un modelo propuesto por *Meta*. Sandonnini (2023) establece que LLaMA 2 destaca por su enfoque audaz en el código abierto, lo que fomenta una comunidad global de investigadores y desarrolladores, mientras que también se enfoca en el aprendizaje reforzado por retroalimentación humana. LLaMA 2 es una buena alternativa si se busca una colaboración entre investigadores.

También, el proyecto LlaMa se ha centrado en mejorar las capacidades de rendimiento de los modelos más pequeños, en lugar de aumentar el número de parámetros (Bergmann, sf.). Así, este enfoque refleja una tendencia hacia el desarrollo de modelos más eficientes, donde se busca maximizar el rendimiento, pero reduciendo costos computacionales y facilitando su implementación.

A pesar de ser una buena alternativa, LLaMA 2 conlleva algunas limitaciones. Touvron (2023) especifica que las pruebas se han realizado en inglés y no han cubierto todos los escenarios en las pruebas realizadas, lo cual conlleva a que los resultados potenciales no se pueden predecir con antelación, y en algunos casos, se pueden generar respuestas inexactas o cuestionables a las indicaciones del usuario. En síntesis, LLaMA 2 es una buena opción, sin embargo, para motivos de la investigación no parece ser el modelo ideal debido a los riesgos que puede presentar debido al contexto abierto que menciona Touvron sobre este.

2. Mistral 7B: Este modelo presenta un lenguaje de 7 mil millones de parámetros, diseñado para un rendimiento y una eficiencia superiores, entre sus innovaciones clave se incluyen el uso de la atención de consultas agrupadas y la atención de ventana deslizante (Andrenacci, 2024). A partir de esto, se trata de un modelo que presenta una base sólida y ofrece un buen desempeño en cuanto a resultados se refiere.

   Por otro lado, el modelo incorpora un mecanismo de atención que puede retener las consultas de forma completa para un mejor procesado de estas, lo que junto a un entrenamiento extenso compuesto de fuentes como libros, artículos académicos, páginas web y datos conversacionales, le dan un entendimiento amplio del lenguaje (McDonald et al, 2024). Por lo cual, sus beneficios recaen en su capacidad de retención y manejo de información, no obstante, este modelo también presenta una demanda computacional algo más elevada.

3. Gemma 7B: Gemma es una familia de modelos de lenguaje pertenecientes a Google, su característica es que son modelos ligeros y de código abierto, disponible

en configuraciones de 2 mil millones y 7 mil millones de parámetros para satisfacer una amplia gama de necesidades computacionales (Zia, 2024). Así, Gemma 7B pertenece a esta familia de modelos de lenguaje que destaca por sus códigos abiertos, y es esta variante la cual tiene disponible 7 mil millones de parámetros.

Caballar (2024) plantea que las evaluaciones del rendimiento de Gemma 7B abarcan generación de código, razonamiento de sentido común, comprensión del lenguaje, razonamiento matemático y respuesta a preguntas. Los resultados indican que es comparable con Llama 3 8B y Mistral 7B, siendo levemente superior a aquellas. Esto puede significar una ventaja frente a los modelos previamente revisados.

A pesar de ser un modelo que presenta ciertas ventajas para este experimento, también hay que considerar algunas limitaciones. Según comenta Gemma Team de Google Deepmind, se reporta cierta sensibilidad al formato de las consultas, marcado en la variante 2B y existente en la variante 7B (2024). Lo anterior puede introducir inestabilidad en tareas que requieren respuestas de estructura consistente, lo que si bien es un problema general de este tipo de modelos de texto y puede ser solucionado mediante fine tuning, la variante 7B de gemma no se encuentra disponible en la plataforma de entrenamiento utilizada.

4. Tiny Llama: Tiny Llama se trata de un proyecto de código abierto que entrena un pequeño modelo de lenguaje con aproximadamente 1100 millones de parámetros, buscando un modelo de lenguaje capaz de realizar tareas que un LLM completo como Llama 2 puede realizar, pero con un menor consumo de memoria (Kjosbakken, 2024). De esta forma, Tiny Llama tiene la capacidad de desarrollar tareas de forma asertiva, sin embargo, a pesar de haber demostrado un buen desempeño, su menor consumo de memoria al ser un modelo con una menor cantidad de parámetros en comparación, puede ser un limitante para el desarrollo de la investigación.

Aún así, el modelo Tiny Llama es un modelo que no presenta una gran variedad de revisiones publicadas en la web, se ha encontrado una menor cantidad de información en comparación a los demás modelos y limitada información de un sometimiento a pruebas. Esta falta de información podría significar la falta de expectativas claras para el desarrollo de la investigación.

5. Phi 3 Mini: Este es un modelo compacto de 3.8B diseñado para la eficiencia computacional, que con una capacidad 128k tokens, representación de la capacidad de memoria del modelo, puede manejar contextos extensos y tareas que requieran un entendimiento de estructuras amplias, a pesar de su reducido tamaño, demuestra un rendimiento competitivo, siendo apto en entornos de recursos acotados (Dong, 2025). Su rendimiento significa una ventaja en comparación a los demás modelos, siendo accesible para el desarrollo de investigaciones de forma universal.

   Se reconoce que Phi 3 Mini ha demostrado ser un modelo ligero pero eficaz en tareas que no requieren un conocimiento factual elevado (Abdin et al, 2024). Lo anterior, sumado a su tamaño compacto, resulta en un buen candidato que no requiere de una potencia computacional elevada, al no necesitar que el modelo reconozca contextos tan extensos para este experimento.

   Dong (2024) también comenta que un incremento en la complejidad de la tarea afecta la calidad de las respuestas de forma substancial. Si bien esto puede significar una limitación, la naturaleza de las pruebas realizadas en esta investigación no conllevan una complejidad elevada mayor, lo que hasta cierto punto, permitiría el uso de este modelo.

4.2 Métodos de generación procedural

En el ámbito de la generación de contenido en videojuegos, los algoritmos procedurales son de los métodos más usados en diversos aspectos como estructuras de mapas, decoración, entre otros. Es por esto, que se consideran algunos métodos conocidos en el

desarrollo de videojuegos para esta investigación. A continuación, se desarrollará una contextualización sobre Cellular Automata, Perlin Noise y Random Walker.

1. Autómata Celular: Es un sistema dinámico discreto que trata de una red regular de autómatas de estados finitos en forma de celda, los que cambian sus estados dependiendo de los estados de sus vecinos de acuerdo con una regla de actualización local, también, son modelos matemáticos para la computación masivamente paralela (Kari, 2024). Según lo expuesto, un Autómata Celular es útil para modelar fenómenos complejos que emergen a partir de interacciones locales simples, lo que permite analizar y simular comportamientos en distintos campos del conocimiento, tales como la física, la biología, la informática y las ciencias sociales.

2. Perlin Noise: Este método utiliza terminología de la física del sonido para definir sus cuatro parámetros: periodos, lacunaridad, persistencia y octavas. Una posible desventaja, es que especificar el conjunto adecuado de parámetros puede resultar confuso, especialmente cuando estos interactúan de forma poco intuitiva. Por lo tanto, la aplicación exitosa de este método de generación procedimental puede requerir mucha experimentación (Etherington, 2022). De acuerdo a esto, este método es muy útil para terrenos o biomas, pero menos eficiente en control topológico o detalles de menor escala.

3. Random Walker (Variante Drunkard Walker): Se refiere a la generación secuencial de celdas conectadas mediante movimientos pseudo aleatorios controlados, que pueden seguir ciertas reglas exploratorias. En este caso, se evaluará la variante Drunkard Walker (Baron, 2017), caracterizada por un desplazamiento aleatorio continuo que garantiza la conexión entre celdas.

5. Evaluación de alternativas

A partir de los modelos de lenguaje previamente revisados, se puede establecer que muchos de ellos presentan ventajas similares y algunas limitaciones particulares. Es necesario considerar estos elementos para establecer el modelo pertinente según las necesidades de la investigación.

Para efectos del presente trabajo, los modelos de lenguaje Phi 3 Mini y Mistral resultan ser opciones adecuadas de acuerdo a sus características, teniendo tanto un contexto como un coste computacional intermedio. Por el contrario, Tiny Llama y Gemma no son consideradas en primera opción. Las limitaciones anteriormente presentadas de Tiny Llama con su capacidad menor, y Gemma con mayor coste computacional podrían influir en los resultados, desarrollando un contexto muy limitado en el caso de Tiny Llama, o una confiabilidad menos certera en respuestas estructuradas, además de un aumento de requerimientos en el caso de Gemma.

Por consiguiente, Phi 3 Mini resulta superior al modelo Mistral en este contexto, puesto que, si bien Mistral es comparable o en ocasiones, superior a Phi 3 Mini en rendimiento debido a una mayor cantidad de parámetros, esto conlleva a una carga computacional mucho más elevada que si bien no llega al coste del modelo Gemma, resulta mayor en comparación a la requerida por Phi 3 Mini. Es por esto, junto a no necesitar de un razonamiento profundo debido al proceso de fine tuning involucrado posteriormente, que Phi 3 Mini resulta ser el modelo escogido para las pruebas a realizar, considerando el contexto y las características del entorno para el desarrollo de esta investigación.

Por otra parte, considerando los algoritmos de generación procedural revisados, se detectan diferentes niveles de complejidad y adaptabilidad, y asimismo, tienen usos en diferentes áreas y para diferentes resultados en el desarrollo de un videojuego. De aquellos, es deseable un algoritmo versátil que permita adaptar sus condiciones de generación, y que a su vez, mantenga una complejidad de implementación moderada. Random Walker resulta

conveniente para realizar pruebas comparativas, debido a las modificaciones que pueden aplicarse a la lógica de exploración del agente de este algoritmo, mientras se mantiene una estructura consistente en las generaciones, y por consiguiente, en las condiciones a verificar.

Finalmente, para las necesidades de comparación de este proyecto no se requiere de un videojuego completo al comparar los patrones de generación que pueden producir ambos métodos. Por esta razón, se opta por utilizar un algoritmo de generación procedural Random Walker, en específico su variante Drunkard Walker, basado en los patrones de generación del videojuego Spelunky, mas no se hace uso del videojuego como tal.

6. Desarrollo del Proyecto

6.1. Definición de la solución

La solución propuesta consiste en un sistema comparativo compuesto por dos generadores, en donde ambos métodos, algoritmo procedural y modelo generativo, son evaluados bajo las mismas restricciones y condiciones de generación:

- Generación de matrices cuadradas, con tamaños entre 4x4 y 9x9 para simular un escenario 2D como el mostrado en la imagen 1, con el fin de mantener control sobre las dimensiones evaluadas. Estas matrices representan un ambiente 2D, similar al nivel de un videojuego.

```
# iter=276, tries=1, valid=True, size=8x8
# matrix (normalized to h x w):
5,5,5,5,5,5,5,2
5,5,5,5,5,5,2,3
5,5,5,5,5,2,3,5
5,5,5,5,2,3,5,5
5,5,5,5,4,5,5,5
5,5,5,5,4,5,5,5
5,5,5,5,3,2,5,5
5,5,5,5,5,3,5,5
```

Imagen 1: Estructura general de las matrices codificadas, obtenida en los experimentos con el modelo de texto.

- Codificación de los enteros que componen la matriz, en representación de posibles cuartos en el mapeado de un videojuego.

    1. Conexión horizontal (Izquierda-Derecha)
    2. Conexión Izquierda-Derecha-Abajo
    3. Conexión Izquierda-Derecha-Arriba
    4. Conexión Completa (4 direcciones)

5. Celda cerrada, equivalente a un muro, o un espacio accesible bajo requerimientos en el juego y que no es parte del camino principal.

- El mapa debe presentar un camino válido que conecte la parte superior de la matriz con la fila inferior, simulando un videojuego de exploración de cavernas como lo es Spelunky. La imagen 2 destaca un camino correcto según la codificación usada.

```
# iter=276, tries=1, valid=True, size=8x8
# matrix (normalized to h x w):
5,5,5,5,5,5,5,2
5,5,5,5,5,5,2,3
5,5,5,5,5,2,3,5
5,5,5,5,2,3,5,5
5,5,5,5,4,5,5,5
5,5,5,5,4,5,5,5
5,5,5,5,3,2,5,5
5,5,5,5,5,3,5,5
```

Imagen 2: Visualización de un camino válido en una matriz.

- Para que la conexión entre dos celdas sea válida, la salida de una celda debe estar conectada directamente a la entrada de otra.

El sistema de comparación ejecuta ambos generadores bajo condiciones idénticas, recolectando métricas comunes sobre desempeño, diversidad y consistencia.

Estas condiciones buscan generar una estructura de mapeado 2D basado en la que podría ofrecer un videojuego, en específico el mencionado *Spelunky (2012)* o estructuras similares como el videojuego *The binding of Isaac (2011)*, manteniendo así una similitud con lo que podría ser un caso de uso real.

6.2. Implementación del Generador procedural

El generador procedural se basó en una variación del algoritmo Random Walker; Drunkard Walker, que hace uso de un agente, llamado agente walker o caminante, para recorrer la

matriz de forma pseudoaleatoria a la vez que va abriendo conexiones válidas bajo reglas predefinidas, habiendo realizado modificaciones extra al algoritmo base para ser aplicado a las pruebas pertinentes (BlackthornProd, 2018).

El algoritmo se inicia con el agente walker en la fila superior de la matriz y avanza hacia la inferior, asegurando al menos un camino continuo hasta la última fila, siendo este camino el recorrido del agente, al tiempo que podría llegar a generar ramificaciones opcionales controladas por probabilidad.

Las imágenes 3 y 4 muestran un posible mapa 2D del estilo Spelunky bajo la generación procedural usada, y una vista interna que destaca el camino principal de este.

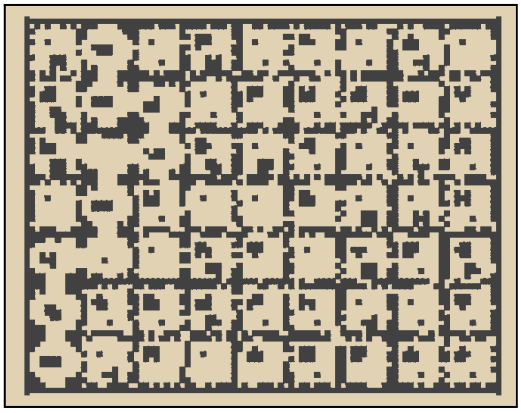

Imagen 3: Simulación de mapa estilo Spelunky basado en el algoritmo Random Walker generado en Unity.

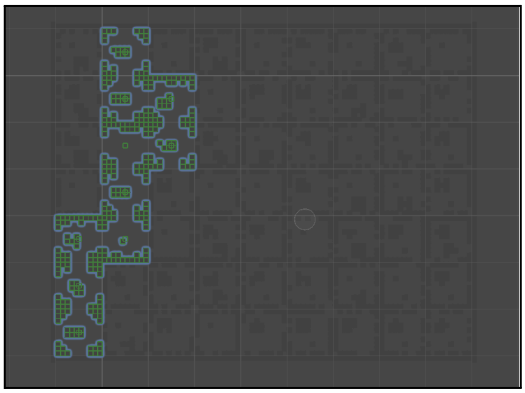

Imagen 4: Vista interna del mapa generado proceduralmente, que demuestra el camino principal.

6.3. Implementación del Generador LLM

El modelo generativo base utilizado se trata del modelo de lenguaje de gran escala Phi-3-Mini, el cual fue considerado por ser un modelo de coste y capacidad intermedios. Phi-3-Mini se identifica como un modelo relativamente pequeño, pero viable para tareas sencillas y acotadas, además de poderse ajustar a tareas específicas con facilidad (Beaty,

2024). La relevancia de aquello recae en la posibilidad de buscar una posible alternativa o complemento a los algoritmos actuales, en los cuales, un coste de procesamiento elevado podría resultar contraproducente.

El modelo fue ejecutado y entrenado localmente en el ambiente de entrenamiento y fine-tuning ofrecido por las librerías de la plataforma web *unsloth.*

Esta implementación considera al modelo generativo, que en su estado base puede responder de forma general y a veces errónea a las consultas relacionadas con este tipo de generación. El modelo fue sometido a un proceso de fine-tuning, orientado a que el LLM pueda asimilar los patrones esperados y así, respetar las condiciones en su generación. Durante el proceso de fine tuning el modelo se entrenó con ejemplos de matrices válidas generadas proceduralmente para aprender la estructura y restricciones de problema de manera contextual. Las imágenes 5, 6 y 7 muestran parte del proceso de entrenamiento realizado, iniciando con la carga del modelo base a entrenar.

```
from unsloth import FastLanguageModel

model_name = "unsloth/Phi-3-mini-4k-instruct-bnb-4bit"

max_seq_length = 2048  # Choose sequence length
dtype = None  # Auto detection

# Load model and tokenizer
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name=model_name,
    max_seq_length=max_seq_length,
    dtype=dtype,
    load_in_4bit=True,

    local_files_only=True,
    device_map={"": 0},
    fast_inference=False,


)
```

```
==((====))==  Unsloth 2025.10.6: Fast Mistral patching. Transformers: 4.56.2.
   \\   /|    NVIDIA GeForce RTX 2060. Num GPUs = 1. Max memory: 6.0 GB. Platform: Linux.
O^O/ \_/ \    Torch: 2.6.0+cu124. CUDA: 7.5. CUDA Toolkit: 12.4. Triton: 3.2.0
\        /    Bfloat16 = FALSE. FA [Xformers = 0.0.29.post3. FA2 = False]
 "-____-"     Free license: http://github.com/unslothai/unsloth
Unsloth: Fast downloading is enabled - ignore downloading bars which are red colored!
```

Imagen 5: Extracto del código encargado de cargar el modelo sin entrenar, al ambiente de entrenamiento de unsloth.

Dado que las matrices presentan un conjunto reducido de 5 opciones y reglas estrictas de generación, cada una de las muestras adquiere un peso significativo para los

entrenamientos. Aquello, junto a pruebas preliminares en este proceso de fine-tuning permiten observar un comportamiento durante el proceso. Como resultado, un dataset de 1000 matrices generadas bajo el algoritmo de generación procedural Random Walker, resultó suficiente para lograr una tasa de éxito que, si bien no es perfecta, es lo suficientemente elevada para permitir una comparación bajo las condiciones de referencia. En la imagen 6 se muestra un registro del proceso de fine-tuning aplicado al modelo base en el ambiente de *unsloth*.

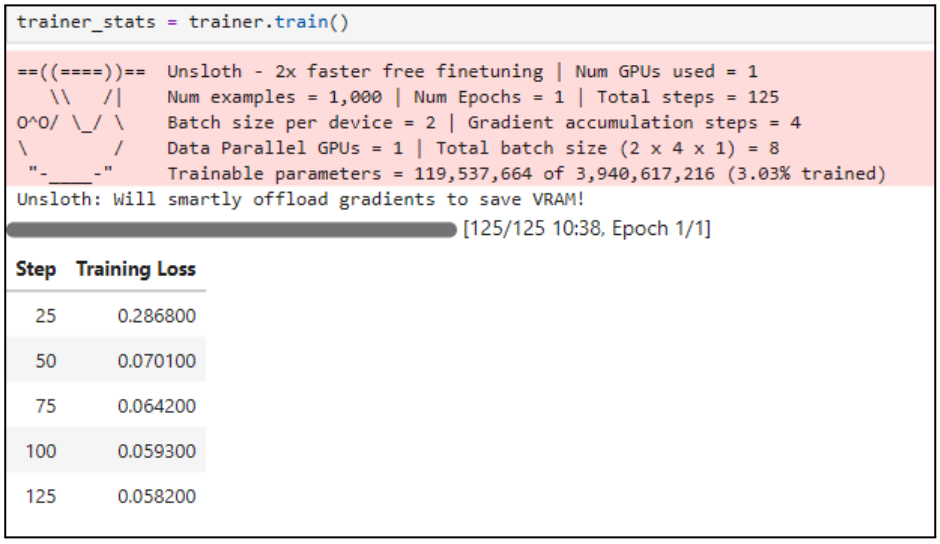

Imagen 6: Extracto del código específico al entrenamiento fine-tuning del modelo.

En la siguiente imagen se muestra una de las posibles respuestas entregadas por el modelo. Esta estructura, donde cada entero y cada fila aparecen separadas, se vuelve algo consistente debido al ajuste del proceso de fine-tuning aplicado al LLM, que logró aprender de manera contextual los patrones de generación del algoritmo procedural al internalizar la estructura del dataset utilizado en el entrenamiento.

```
# raw LLM text (for debugging parse errors):
.
[[5, 5, 5, 2, 5], [5, 5, 5, 3, 2], [5, 5, 5, 2, 3], [5, 5, 5, 4, 5], [5, 5, 5, 3, 5]]
```

Imagen 7: Resultado entregado por el modelo luego del proceso de fine-tuning.

Para cada generación del modelo se usó la siguiente configuración de inferencia (Hiper parámetros), aplicados al modelo de texto:

- Temperatura: Este parámetro determina qué tan restringidas (0) o qué tan abiertas y variadas (1) pueden ser las respuestas del modelo generativo. En este caso, se ha escogido un valor moderado de 0.7, inclinándose por otorgar cierta libertad a las respuestas del modelo. Así, se busca fomentar más variabilidad en las matrices generadas. Por el contrario, valores cercanos a 0 lograrían que el modelo entregue respuestas más deterministas, demasiado similares al dataset de entrenamiento.

- Tokens máximos: Se entiende por tokens a la forma en que un LLM cuenta caracteres, palabras, o combinaciones de palabras y símbolos que generan los LLM cuando descomponen el texto y procesan las consultas internamente (Microsoft, 2025). Para motivos de la investigación se ha determinado un valor relativamente bajo al realizar las pruebas, observando que dentro de 256 tokens una respuesta que incluye una matriz de hasta 9x9 está contenida sin truncarse ni entregar texto extra en la respuesta. Para matrices de mayor capacidad este número puede necesitar ser elevado para evitar truncar la respuesta del modelo al aumentar los caracteres de la respuesta.

- Top-p: Este parámetro determina la variedad de tokens que tiene acceso el modelo al dar una respuesta. Normalmente, un valor reducido que implicaría un vocabulario menor sería deseable para este tipo de pruebas, no obstante, en este caso se opta por estipular un valor elevado de 0.9. Lo anterior se determina en consideración de que el proceso de fine-tuning ya limita el vocabulario utilizado por el modelo a los caracteres y palabras necesarias, por lo cual, un valor elevado otorgaría mayor variedad a las respuestas con bajo riesgo de recibir palabras extra.

- Se opta por semillas de generación aleatoria, puesto que se espera comparar las diferentes respuestas que pueda entregar el modelo de forma general, a diferencia de respuestas predecibles o específicas bajo el uso de una semilla fija.

- El modelo, luego del proceso de fine-tuning, fue exportado bajo una cuantización de precisión 8-bit (Q8). Se entiende la cuantización como una técnica que reduce los parámetros y variables de modelos de alta precisión a otros de menor precisión, es decir, a variables más pequeñas y manejables computacionalmente (Valenzuela, 2024). Así, se logra reducir el peso del modelo aproximadamente a la mitad.

- El resto de la configuración corresponde a la configuración base del modelo utilizado. No se especificaba una penalización de longitud de respuesta al tener una respuesta acotada por los tokens máximos, debido a la obtención de una estructura de respuesta definida, o top k, que simplifica o da más libertad al glosario que puede usar el modelo, que similar a top-p, ya es acotado por el proceso de fine tuning.

La imagen 8 muestra la función generativa utilizada en el algoritmo de comparación, llamando al modelo de texto cargado para realizar la consulta correspondiente y así generar la matriz, lo cual se desarrolla en base a un prompt precargado. Lo anterior se realiza mediante *model.generate(...)*, función que llama al método de generación nativo de los transformers de la plataforma HuggingFace.

```python
def llm_generate_matrix(model, tokenizer, h: int, w: int,
                        temperature=0.7, top_p=0.9, max_new_tokens=256, device=None) -> str:
    prompt = (
        "Encoding:\n"
        "1 = (Left-Right)\n"
        "2 = (Left-Right-Down)\n"
        "3 = (Left-Right-Up)\n"
        "4 = (Left-Right-Up-Down)\n"
        "5 = (no openings)\n\n"
        f"Generate a {h}x{w} matrix"
    )
    messages = [{"role": "user", "content": prompt}]
    inputs = tokenizer.apply_chat_template(messages)

    text = model.generate(
        input_ids=None,
        attention_mask=None,
        max_new_tokens=max_new_tokens,
        temperature=temperature,
        top_p=top_p,
    )
    return text
```

Imagen 8: Extracto del código encargado de cargar el modelo ya entrenado, y generar el texto en base a una solicitud base.

Haciendo uso del proceso descrito y los algoritmos de interés, se han obtenido las métricas necesarias para realizar la comparación entre los dos métodos.

6.4. Métricas de comparación

Mediante un script principal, se automatiza la ejecución de ambos métodos de generación, obteniendo resultados comparativos en forma de tablas de promedios e histogramas de distribución. A continuación, se describen las variables a considerar en la comparación en la tabla 1:

Variable	**Descripción**	**Propósito**
gen_time_s	Tiempo de generación	Eficiencia computacional
rss_mb, vram_mb	Uso de RAM y VRAM	Costo de recursos

path_valid	Condición mínima de un camino válido	Consistencia estructural
fill_density	Porcentaje de celdas activas	Densidad/llenado del mapa 2D
entropy_bits	Entropía de Shannon	Diversidad estructural
turn_rate	Proporción de curvas del camino	Complejidad topológica
attempts	Intentos que necesitó el LLM para lograr una respuesta válida	Estabilidad del modelo generativo

Tabla 1: Nombres de las variables a recopilar para la comparación de los generadores, además de sus definiciones.

6.5. Análisis comparativo

Con el objetivo de comparar los resultados entregados por los métodos de generación procedural y generativo, se ha determinado la realización de dos experimentos. Como primer experimento se evalúa el caso más básico descrito anteriormente, con una lógica sencilla que busca presentar un camino principal para comprobar su factibilidad. Como segundo experimento, se evalúa un caso que complejiza la lógica de generación a múltiples caminos, para comprobar así la capacidad del modelo generativo de aprender diferentes patrones.

En ambos experimentos se desarrollan una serie de iteraciones de generación de matrices con ambos métodos: generativo y procedural. A partir de lo cual, se recopilaron las métricas de comparación descritas y también, promedios representativos.

6.5.1 Primer experimento

En cada iteración se genera inicialmente una matriz cuadrada de dimensión aleatoria mediante un método procedural. Sobre la base de esta matriz, se recopilan las métricas de interés, las cuales eran posteriormente almacenadas para su análisis.

Luego, el prompt utilizado para generar las matrices se enviaba al modelo cargado, utilizando las mismas dimensiones empleadas en la iteración correspondiente del algoritmo procedural, con el objetivo de mantener una consistencia estructural en las comparaciones. Posteriormente, se recibía la respuesta generada por el modelo y, en caso de no cumplir con la condición esperada, el proceso se repetía hasta alcanzar un número máximo de intentos, registrando en cada uno de ellos las métricas asociadas al modelo de texto

6.5.2 Segundo experimento

Posterior a la recopilación y validación de los resultados de la primera comparación, se ha realizado este segundo experimento y posteriormente, una nueva comparación correspondiente. En este caso, se complejiza el algoritmo procedural para así evaluar la retención del modelo expuesto a un patrón menos directo. Esta modificación de la lógica del algoritmo procedural consiste en aplicar múltiples agentes walker en la generación del camino principal de la matriz.

Este experimento se inicia con un solo agente walker de forma similar a la primera prueba. En cada uno de los movimientos existe la posibilidad de generar un nuevo agente en una dirección horizontal contraria, bajo una probabilidad y hasta un límite definido de agentes por matriz. Además, se utiliza una referencia de matrices de mayor dimensión que los de la primera prueba, para así poder observar de mejor forma los patrones de generación de los múltiples agentes, resultado que puede verse en la imagen 9.

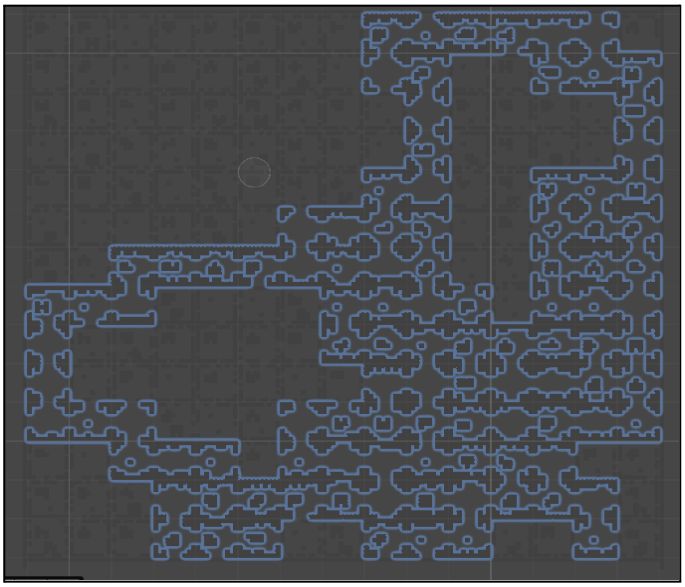

Imagen 9: Captura del camino principal de una matriz, generada por el algoritmo procedural modificado con múltiples agentes.

La configuración utilizada y modificaciones en la generación de múltiples agentes walker corresponden a:

- Un incremento de las dimensiones de las matrices cuadradas a 10, 11, 12, 13, 14 y 15 casillas por lado, suficiente para dar espacio a los agentes a dispersarse.
- 4 Agentes walkers como máximo por matriz, para evitar una sobrepoblación de estos al generar el camino.
- Probabilidad de generación de un nuevo agente walker del 30% en cada movimiento, asegurando que existan bifurcaciones, si bien de manera repartida, cercanas a las filas superiores.
- Se aumentaron los tokens de respuesta de 256 a 900, para compensar el aumento de largo de la respuesta debido al aumento en las dimensiones de las matrices.
- Debido al aumento de posibles ramificaciones y conexiones, se aumentó el dataset de 1000 a 2000 matrices.
- Se entrenó nuevamente el modelo generativo, bajo las condiciones anteriores, ya que estas habían probado dar resultados válidos en el primer experimento.

7. Pruebas y Resultados

7.1. Diseño experimental

Los experimentos tuvieron como propósito comparar el desempeño entre dos enfoques de generación de mapas 2D: un algoritmo procedural de tipo Random Walker (Variación Drunkard Walker) y un LLM ajustado mediante fine-tuning. La experimentación se orientó a evaluar las diferencias cuantitativas en tiempo de ejecución, eficiencia en el uso de recursos, diversidad estructural y estabilidad en la generación de resultados válidos del modelo de texto.

Para asegurar resultados correctos del estudio, se establecieron condiciones controladas y equivalentes entre ambos métodos, definiendo la misma dimensión de matrices por iteración, al igual que restricciones topológicas de las celdas y la conectividad vertical requerida. En el caso del LLM, las respuestas fueron validadas según la lógica del algoritmo random walker, asegurando que las respuestas entregadas por este se evalúan según las mismas condiciones estructurales.

Para ambos experimentos, se ejecutaron un total de 300 repeticiones independientes por método generativo, número en el que métricas de interés mostraban suficiente convergencia, con el fin de obtener una muestra estadística representativa. En cada iteración se registraron los valores de las métricas definidas:

- Tiempo de generación
- Uso de memoria RAM
- Uso de VRAM (LLM)
- Densidad de llenado de celdas
- Entropía estructural
- Tasa de curvatura del camino
- Intentos hasta una respuesta válida (LLM)

Posteriormente, se calcularon promedios para cada métrica, permitiendo comparar la consistencia y variabilidad entre ambos enfoques.

Asimismo, se realizó un análisis de distribución y aleatoriedad utilizando pruebas de Chi^2 y divergencia de Kullback-Leibler (KL) como candidatos de estas métricas. La elección del tipo de pruebas se debe a que estas calculan medidas de distribuciones agregadas, en lugar de muestras específicas. Además, tienen como objetivo cuantificar el grado de similitud estructural entre las distribuciones generadas por el modelo LLM y el método procedural, no solamente su similitud directa.

El interés por la divergencia de Kullback-Leibler reside en su capacidad de medir la pérdida de información que ocurre cuando la distribución producida por el LLM se aproxima o aleja de la distribución del modelo general. Mientras que el interés de las métricas ofrecidas por Chi^2 radica en que resulta un cálculo más sensible a las variaciones de distribución, ofreciendo un punto de comparación a la distribución KL.

El experimento fue ejecutado en un entorno local, garantizando condiciones uniformes de hardware y software, evitando dependencias externas o diferentes que pudieran alterar los resultados. Con ello, se logró una evaluación objetiva y replicable de ambos métodos bajo un mismo marco de prueba.

7.2 Resultados y Análisis

7.2.1 Primer experimento

En el primer experimento, se contrastaron los resultados de las matrices generadas con un único Agente Walker, frente a las producidas por las matrices del modelo procedural. Los resultados obtenidos se presentan a continuación:

Métrica	Procedural (Random Walker)	LLM (fine tuned phi-3-mini)
Tiempo promedio (s)	0.0001	2.3782
Uso de RAM (MB)	0.0009	1228.9
Uso de VRAM (MB)	0 (solo procesador)	5925.25
Densidad de celdas del camino (proporción)	0.256	0.2055
Entropía (bits)	1.2631	1.0372
Curvatura (proporción)	0.405	0.2573
Intentos Promedio	1	1.8066

Tabla 2: Tabla comparativa de promedios de los resultados obtenidos experimentalmente con cada método generativo, con un walker creando el camino principal.

Basado en los resultados expuestos en la tabla 2, es que se pueden obtener las siguientes observaciones:

- Velocidad: El algoritmo procedural supera al LLM en 4 órdenes de magnitud, demostrando ser más ligero en ejecución al momento de generar las matrices usadas para la estructura del mapa 2D.

- Uso de recursos (RAM y VRAM): Como cabría esperar, el algoritmo procedural presenta un consumo significativamente menor al operar directamente en el procesador, sin requerimientos mayores. En contraste, el modelo de texto demanda un uso considerablemente mayor en ambos puntos, requiriendo una cantidad

significativa de capacidad gráfica (VRAM) para cargar el modelo, y mayor uso de memoria RAM en los procesos de inferencia que el algoritmo procedural.

- Densidad: El algoritmo procedural muestra una proporción de densidad (celdas del camino principal vs celdas totales de la matriz) ligeramente mayor que la proporción entregada por el modelo de texto, lo que por sí solo podría ser aceptable y útil en el contexto de un mapa 2D de estas características. Experimentalmente, posibles causas de esto podría deberse a la necesidad de un mayor set de datos de entrenamiento, la cuantización del modelo luego del fine-tuning, o capacidades computacionales locales.

- Estabilidad: El algoritmo procedural, al poseer una lógica de condiciones y objetivo cerrados, presenta caminos válidos el 100% de las veces. El LLM, si bien logra resultados correctos en la mayoría de casos, puede llegar a necesitar generalmente entre 1 o 2 intentos con la configuración dada, para entregar una matriz válida.

- Entropía y curvatura: Nuevamente, el algoritmo procedural presenta un valor de diversidad ligeramente más alto que las generaciones del modelo. Esto puede deberse al entrenamiento contextual del modelo, reproduciendo estructuras similares que potencian el cumplir la condición de término por sobre las posibles curvas en la matriz, comparado a la lógica probabilística establecida en el algoritmo procedural. No obstante, los valores resultan lo suficientemente similares para considerar la diferencia como algo válido.

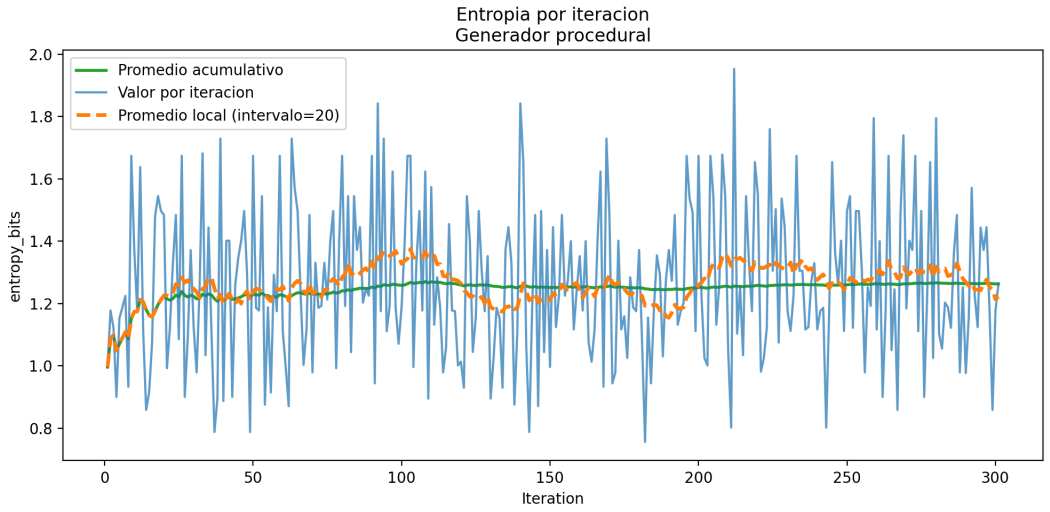

Gráfico 1: Métricas de entropía y promedios obtenidos a partir de las generaciones procedurales.

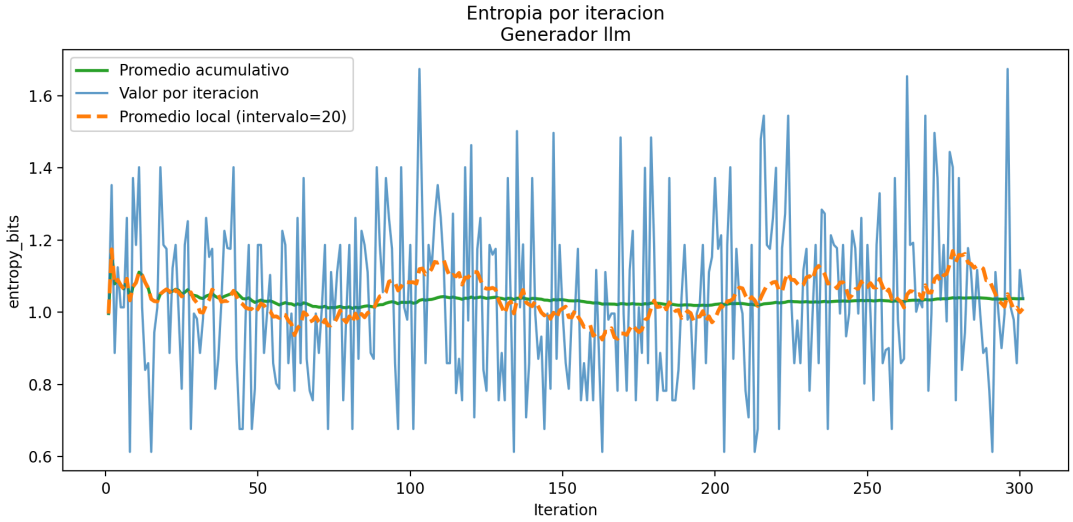

Gráfico 2: Métricas de entropía y promedios obtenidos a partir del modelo generativo.

Observando los gráficos 1 y 2 sobre entropía entre ambos métodos de generación, se puede ver que se presenta un promedio lineal, correspondiente con una distribución equilibrada al tomar en cuenta las múltiples iteraciones

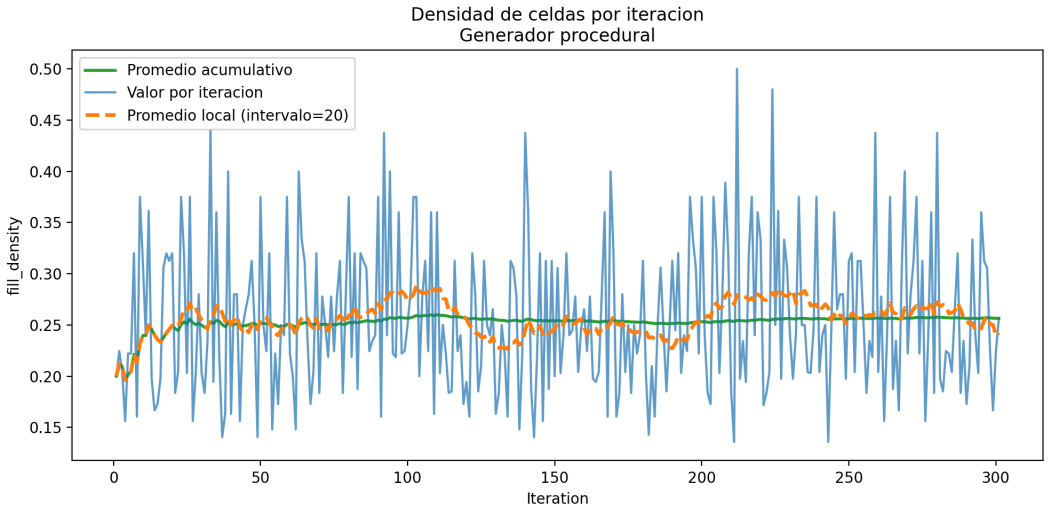

Gráfico 3: Métricas de densidad de llenado de celdas y promedios obtenidos a partir del algoritmo procedural.

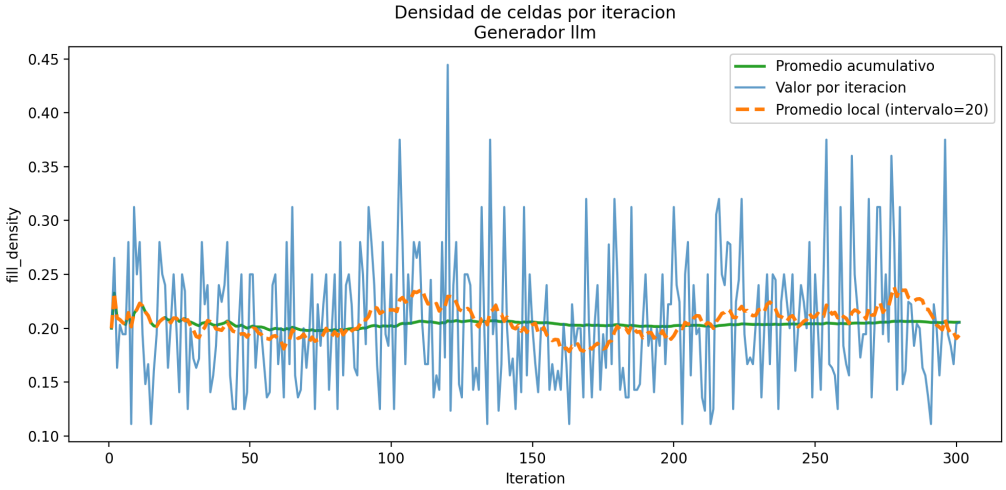

Gráfico 4: Métricas de densidad de llenado de las celdas y promedios obtenidos a partir del modelo generativo.

En los gráficos 3 y 4 nuevamente se puede observar este equilibrio en el promedio de los datos, ocurriendo desde las generaciones iniciales y manteniéndose a lo largo de la prueba.

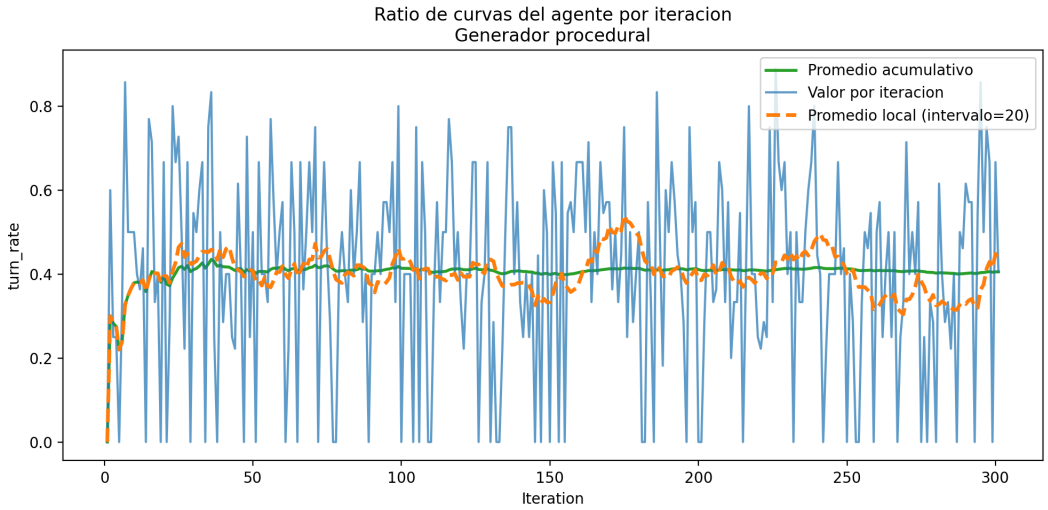

Gráfico 5: Métricas de curvatura del agente y promedios obtenidos a partir del algoritmo procedural.

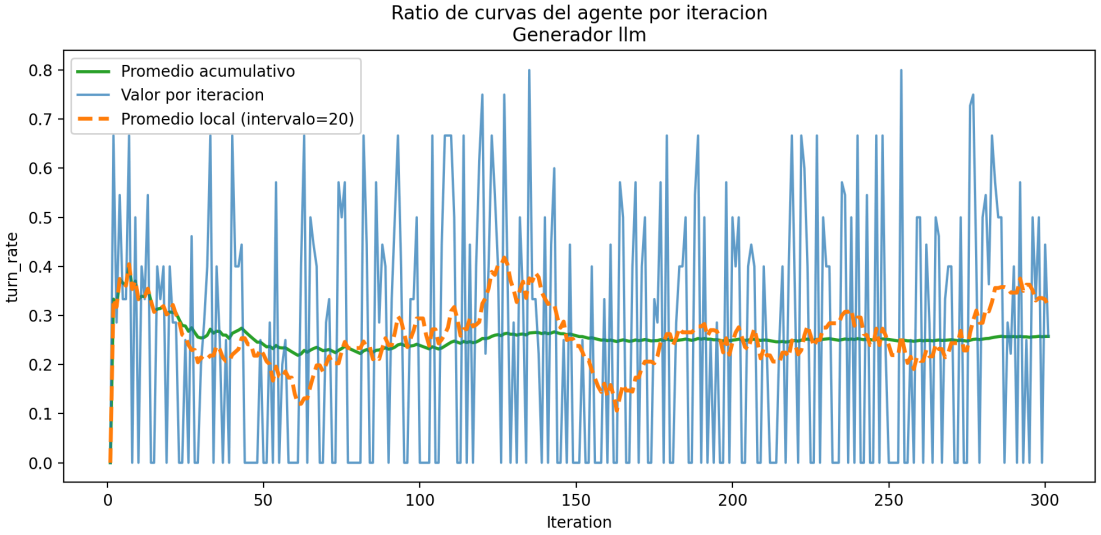

Gráfico 6: Métricas de curvatura del agente y promedios obtenidos a partir del modelo generativo.

Luego, al analizar los gráficos 5 y 6, se establece que si bien el equilibrio en el promedio acumulativo se mantiene desde iteraciones tempranas, se puede observar una mayor cantidad de picos locales para esta variable y que son ligeramente más notorios en las generaciones del modelo generativo. Lo anterior sugiere una variación que se inclina a ciertos resultados positivos que pueden haber sido favorecidos durante el fine tuning del modelo. Al evaluar estos datos, se puede observar que en varios casos el agente bajó en línea recta por la matriz, sugerido por los picos locales bajo el promedio.

Observando los seis gráficos correspondientes al primer experimento sobre entropía, densidad de llenado y proporción de curvatura de los agentes, se puede determinar que las métricas se presentan de forma altamente variada, lo que coincide con un dataset de matrices generadas de forma aleatoria.

Además, al evaluar los promedios acumulativos y locales, se puede determinar un comportamiento equilibrado en las variables dentro de 300 iteraciones. Asimismo, al comparar los valores entre cada método de generación, el equilibrio se presenta en valores similares, lo que apoya la similitud entre los patrones de generación de ambos métodos. Esta linealidad general resulta algo deseable, ya que descarta que los patrones de generación del modelo generativo favorezcan valores elevados o reducidos de estas métricas, adhiriéndose a la referencia que entrega las métricas del generador procedural.

	Distribución de celdas	Ratio de curvas
Chi^2	1469.238	100000147.166
Kullback–Leibler (KL)	0.036	0.269

Tabla 3: Medidas de divergencia usadas para comparar la "distancia" entre las distribuciones de ambos métodos.

Por último, de acuerdo a los resultados mostrados en la tabla 3, se puede observar que el valor obtenido de KL para la distribución de celdas se encuentra cercano a 0, lo que indica que, a pesar de existir diferencias entre ambos métodos, las distribuciones estructurales que estos generan son similares. Luego, el valor de divergencia KL para el ratio de curvas entre los modelos es algo mayor, indicando una posible diferencia, lo que está de acuerdo con los resultados de la tabla 2. Aquellos resultados indican que el algoritmo procedural posee una proporción ligeramente mayor de curvas que las generaciones hechas por el modelo, las que presentan caminos a veces más directos y que podrían quedar sujetos a entrenamientos más extensos, menor sobreajuste, o un set de datos más variado para llegar a una distribución estructural más cercana.

Sin embargo, los valores para la distribución Chi^2 aumentan significativamente, mostrando una clara diferencia en distribución y ratio de curvas, puesto que este cálculo de distribuciones contrasta las diferencias absolutas entre ambos generadores. No obstante, el objetivo de este experimento trata de evaluar si los resultados entregados por el modelo de texto son viables estructuralmente, no una copia directa de los generados por el algoritmo procedural (dentro de una misma iteración, las matrices de ambos generadores son aleatorias). Por lo tanto, la métrica finalmente no resulta representativa en este experimento al ser demasiado sensible a los cambios de distribución estructural.

7.2.2 Segundo experimento

Debido a que los resultados obtenidos de las comparaciones aplicadas a ambos métodos según la lógica de generación de un agente Walker, es que se consideró verificar el desempeño del modelo generativo bajo una lógica con una complejidad mayor. En este caso, se utilizan Múltiples Walkers.

Métrica	Procedural (Random Walker)	LLM (fine tuned phi-3-mini)
Tiempo promedio (s)	0.0006	10.3042
Uso de RAM (MB)	0.0152	1242.5151
Uso de VRAM (MB)	0 (solo procesador)	5790.3981
Densidad de celdas del camino (proporción)	0.4368	0.1427
Entropía (bits)	1.8278	0.8368
Curvatura (proporción)	0.455	0.5659
Intentos Promedio	1	12.12 (8.2388)
Ratio de validez	100%	66.78%

Tabla 4: Tabla comparativa de promedios de los resultados obtenidos experimentalmente con cada método generativo, con matrices de mayor complejidad y un parámetro añadido.

A partir de la tabla 4, se puede determinar que el algoritmo procedural presentó un aumento muy ligero de coste comparado a los resultados de la tabla 2, debido a la complejidad del algoritmo procedural con múltiples agentes walker. No obstante, este aumento de coste es despreciable debido al orden de magnitud en el coste de tiempo. Si bien se observa una elevación de dos órdenes de magnitud en el coste de memoria, aún, no llega a acercarse al coste de memoria del modelo generativo.

En el caso del modelo generativo, si bien su coste de memoria RAM y memoria gráfica (VRAM) siguen siendo elevados, estos se mantuvieron similares al coste que estos presentaron en las pruebas de un solo agente. No así el coste temporal, que en promedio

aumentó 5 veces en comparación a las pruebas anteriores, aunque gran parte de esto no se debe al aumento de complejidad, si no al aumento de dimensiones, que incurren en un aumento directo en los tokens que el modelo generativo debe procesar en cada respuesta y aporta el mayor aumento de coste temporal.

En este caso se presenta un problema, a diferencia de la prueba anterior, que presentaba un índice de validez del 100% junto a su algoritmo procedural, mientras que el modelo generativo basado en patrones de múltiples agentes walker presenta un índice de validez del 66%, y un aumento de intentos promedio del 1.8 al 8.2 al aumentar la complejidad (obviando iteraciones en los que 20 intentos fueron superados). Esto muestra que un aumento de complejidad en los patrones estructurales de un mapeado 2D resulta en un desafío para un modelo generativo.

Junto a esto, se determinó que existe una bajada en los índices de densidad correspondiente al camino principal y la entropía de las matrices del modelo generativo. Esto es una muestra de que el modelo pudo haber aprendido patrones más conservadores y directos en su generación.

	Distribución de celdas	Ratio de curvas
Chi^2	34460.88	4000000162.77
Kullback–Leibler (KL)	0.28	1.11

Tabla 5: Medidas de divergencia aplicadas a las matrices del segundo experimento, para comparar la "distancia" entre las distribuciones de ambos métodos. Solo consideran en el cálculo a las matrices válidas.

Finalmente, según los resultados de distribución estructural mostrados en la tabla 5, se observa nuevamente que las métricas de Chi^2 son altamente sensibles, puesto que destaca diferencias muy leves entre los valores entregados por ambos métodos de generación. Esta sensibilidad podría llevar a interpretar variaciones que, en la práctica, no representan diferencias significativas entre los métodos de generación. Por lo tanto, los resultados

deben ser analizados con cautela y complementados con otros indicadores, ya que, en vista de los resultados, se observa que la sensibilidad de la métrica Chi^2 por sí sola requeriría una distribución estructural casi idéntica para denotar mayor similitud, lo que no es el objetivo de esta comparación.

Luego, para las métricas correspondientes a la divergencia de Kullback-Leibler, se puede observar que para este experimento, el valor de la divergencia para la distribución de celdas aumentó, sugiriendo que la distribución estructural entre las matrices válidas generadas por el modelo generativo comparadas a las generadas por el algoritmo procedural, se vuelve mayor y por tanto se diferencian. Aún así, este aumento tiene sentido, debido a que la inclusión de más agentes walker y por tanto, bifurcaciones en la generación, aumenta la diversidad que entregan ambos métodos.

Por último, el valor de KL para el ratio de curvas presentadas en las matrices disminuyó, esto indica un acercamiento en las distribuciones de las matrices de ambos métodos. Así, los resultados son coherentes con el aumento de complejidad y de caminos presentes en las matrices de este segundo experimento al existir mayores posibilidades de coincidir debido a los múltiples agentes walker y sus respectivos caminos, dando al modelo generativo mayor probabilidad de coincidir con alguno de estos.

8. Análisis de resultados

A partir de los procedimientos realizados y los resultados experimentales obtenidos, se puede determinar que el modelo de lenguaje de gran escala *phi-3-mini,* ajustado mediante fine-tuning, logró generar mapas válidos y estructuralmente coherentes bajo los criterios y lógica especificados, demostrando potencial aplicabilidad en tareas de mapeado 2D.

El análisis comparativo entre ambos métodos revela que los resultados presentan una similitud topológica satisfactoria, demostrando que la IA generativa puede llegar a reproducir estructuras espaciales con coherencia lógica y consistencia. No obstante, también existen diferencias claras en eficiencia y consumo de recursos. De esta manera, el enfoque procedural presentó una ventaja significativa en tiempo de ejecución y requerimientos de hardware, mientras que el modelo de texto demandó una cantidad importante de RAM y VRAM. Esto confirma que, si bien ofrece flexibilidad generativa, su costo operativo limita su aplicación a contextos experimentales o de prototipado, o bien ser ofrecido en forma de *Model as a Service (*Modelo como Servicio, MaaS), donde el consumo de recursos no recae de forma directa sobre el usuario final.

En cuanto a la diversidad estructural, las métricas de entropía y de curvatura indican que el algoritmo procedural genera resultados más variados y con trayectorias más irregulares. Por su parte, el modelo generativo, bajo las condiciones y entrenamiento realizado en este experimento, tiende a configuraciones más simples y directas. Este comportamiento, junto con la disminución en la tasa de validez y el aumento en el número de intentos observados en el segundo experimento, sugiere una sensibilidad del modelo frente al aumento de la complejidad estructural.

Asimismo, la menor entropía y curvatura indican que el modelo tiende a favorecer configuraciones más frecuentes o estables dentro del contexto asimilado por el modelo. No obstante, esta tendencia no implica una correcta internalización de las restricciones del problema, particularmente las condiciones de conectividad y final del camino, lo que se traduce en una menor proporción de soluciones válidas pese a la aparente simplicidad de

las estructuras generadas. Este efecto se vuelve más evidente al aumentar la complejidad del escenario, donde dicha preferencia por configuraciones estables resulta insuficiente para capturar patrones más exigentes. Una opción no explorada en este experimento y que podría suplir esta deficiencia es el uso de modelos más especializados o de mayor capacidad, los cuales podrían ofrecer un mejor desempeño dada la amplia variedad de modelos disponibles actualmente.

Pese a las diferencias, los resultados obtenidos se mantienen dentro de márgenes aceptables. Esto respalda la capacidad del modelo de texto para reproducir los patrones estructurales del algoritmo procedural, junto a la posibilidad de seguir siendo entrenado y ajustado a diferentes patrones de mapeado 2D. De esta manera, se podría aproximar aún más al desempeño de un algoritmo procedural.

Desde una perspectiva estadística, los valores de divergencia KL cercanos a cero en la distribución de celdas, reafirman que el modelo generativo logra replicar, o al menos aproximarse de forma consistente, a la lógica espacial del método procedural con precisión. Si bien el test Chi^2 arrojó valores elevados, esto se explica principalmente por la aleatoriedad utilizada en ambos sistemas, lo cual no le resta validez a los resultados del experimento, ya que el objetivo principal fue evaluar la similitud estructural del modelo generativo con el modelo generativo y no la obtención de una correspondencia exacta entre ambas metodologías.

En términos de estabilidad, se asume que el generador procedural siempre cumple las condiciones debido al algoritmo usado, por lo que el mayor interés recae en el modelo generativo. Es posible observar que el LLM requirió en promedio entre uno o dos intentos para producir una matriz correcta, generalmente necesitando una regeneración de la respuesta. Este comportamiento refleja una necesidad de mayor precisión en los resultados entregados por el modelo generativo, ya que si bien los resultados correctos cumplen mayormente con los criterios de forma satisfactoria, no es ideal que aproximadamente el 40% de los resultados del modelo generativo deban ser repetidos. Este aspecto podría ser optimizado mediante un conjunto de entrenamiento más amplio y variado, o a través de un

ajuste más fino de los hiper parámetros del modelo, tanto en la etapa de entrenamiento como en la de generación final.

Este fenómeno se vuelve aún más crítico al aumentar la complejidad del mapeado, como se evidenció en el segundo experimento, donde el modelo presentó mayores dificultades para retener patrones más complejos derivados del uso de múltiples agentes *walker* en el algoritmo procedural en la generación del camino. No obstante, pese que la tasa de validez no alcanzó el 100% durante los experimentos en el mapeado más complejos, los resultados válidos continúan mostrando distribuciones estructurales deseables, lo que reafirma la posibilidad de que un modelo generativo pueda llegar a ser un complemento, dado el caso de disponerse de un modelo de mayor capacidad, presentar un dataset que fomente mejor los entrenamientos del modelo, y/o de ajustar hiper parámetros de diferente manera.

9. Conclusiones

Considerando el objetivo de la investigación, se puede determinar que los resultados permiten afirmar que la inteligencia artificial generativa, bajo un proceso adecuado de *fine-tuning* y dependiendo del modelo generativo, puede constituir una alternativa viable y complementaria a la generación procedural tradicional. Su mayor fortaleza radica en la capacidad de adaptación y aprendizaje contextual a partir de un conjunto de datos establecido, lo cual abre nuevas posibilidades en la generación de contenido.

Sin embargo, también se presentan desafíos relevantes, específicamente en términos de eficiencia y dependencia de recursos computacionales. Este elemento continúa siendo una limitación importante no solo para entornos de desarrollo de videojuegos en donde el coste computacional ya suele ser elevado, sino que también, se ve afectada la implementación práctica de modelos generativos en diversas áreas de la computación.

A pesar de estas limitaciones, el estudio demuestra que los modelos generativos pueden aproximarse de manera consistente a la lógica estructural de métodos algorítmicos clásicos, especialmente cuando se evalúan métricas de distribución y coherencia espacial. Lo anterior sugiere que con modelos de mayor capacidad, conjuntos de entrenamiento más representativos y una optimización ideal de los hiper parámetros, el rendimiento del enfoque generativo podría mejorar de forma significativa.

Finalmente, este estudio contribuye al conocimiento sobre la integración de modelos generativos en la creación de contenido para videojuegos, evidenciando que, aunque los métodos de algoritmos clásicos continúan siendo más eficientes y confiables, los enfoques basados en IA generativa ofrecen nuevas oportunidades para el diseño automatizado y exploración de soluciones emergentes.

Como trabajo futuro, se recomienda ampliar y diversificar el conjunto de entrenamiento, añadir obstáculos u objetos de interés a los patrones de generación, explorar modelos de

mayor capacidad o especializados, y evaluar su desempeño en entornos tridimensionales o en escenarios híbridos que combinan ambos enfoques.

Referencias

Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Ahmad, A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., Benhaim, A., Bilenko,M., Bjorck, J., Bubeck, S., Cai, M., Cai,Q., Chaudhary, V., Chen, D., Chen,D., Chen W (...). (2024). *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone.* Microsoft. https://arxiv.org/abs/2404.14219

Andrenacci, G. (09 de julio de 2024). Afinando el Mistral-7B: Un viaje a través de la literatura y la IA. *Medium.* https://medium.com/data-bistrot/fine-tuning-mistral-7b-a-journey-through-literature-and-ai-3322e68b65c3

Baron, J. (2017). *Procedural Dungeon Generation Analysis and Adaptation*. Proceedings of ACM Kennesaw, Clemson University School of Computing. DOI: dx.doi.org/10.1145/3077286.3077566

Bea, J. (2017). *Generación procedimental de contenido en videojuegos*. Universitat de les Illes Balears. España. http://hdl.handle.net/11201/151190

Beauty, S. (2024). *Pequeños pero poderosos: los modelos de lenguaje pequeños de Phi-3 con gran potencial.* Microsoft. https://news.microsoft.com/source/latam/features/ia/pequenos-pero-poderosos-los-modelos-de-lenguaje-pequeno-de-phi-3-con-gran-potencial/

Bergmann, D. (s.f). *¿Qué es llama 2?*. International Business Machines IBM. https://www.ibm.com/es-es/think/topics/llama-2

BlackthornProd. (2018). *Spelunky-Random-LV-GEN.* Github. https://github.com/BlackthornProd/Spelunky-Random-LV-GEN

Caballar, R. (2024). *¿Qué es Google Gemma?*. International Business Machines Corporation. https://www.ibm.com/es-es/think/topics/google-gemma

Centro de Estudios de Innovación, Diseño y Marketing, CEI. (2025). *Diseño de niveles de videojuegos.* CEI. https://cei.es/diseno-niveles-videojuegos/.

Church K., Chen Z y Ma Y. (2021). Emerging trends: A gentle introduction to fine-tuning. *Natural Language Engineering* 27(6):763-778. doi:10.1017/S1351324921000322.

Cuadra, F. (2023). *Técnica para la generación procedural de niveles en juegos tipo roguelike*. Facultad de Informática, Universidad Complutense de Madrid.

Dong, Q. (2025). Code Quality of Open-Source LLM: A Code Smell Analysis [Tesis de Maestría]. Saarland University, Alemania.

Douglas, M (2023). *Large Language Models*. CMSA Universidad de Harvard y Departamento de Física, Universidad de Stony Brooks. https://arxiv.org/pdf/2307.05782

Etherington, T. (2022). Perlin noise as a hierarchical neutral landscape model [Perlin-Noise como modelo de paisaje neutral y jerárquico]. *Web Ecol 22*, 1–6. https://doi.org/10.5194/we-22-1-2022

Farrokhi, M., y Zhao, R. (2024). Procedural Content Generation in Games: A Survey with Insights on Emerging LLMIntegration. *Department of Computer Science, University of Calgary, Canadá.*

Gemma Team, Google Deepmind. (27 de junio de 2024). Gemma 2: Mejorando los modelos de lenguaje abierto a un tamaño práctico. *Google Deepmind.*

Kari, J. (2024). Cellular Automata. University of Turku, Finland.

Kazemi, D. (2013). *Spelunky Generator Lessons*. Tiny Subversions. https://tinysubversions.com/spelunkyGen/

Kjosbakken, E. (20 de abril de 2024). Tiny Llama: una evaluación de desempeño y un debate. *Towards Data Science.* *https://towardsdatascience.com/tiny-llama-a-performance-review-and-discussion-a68d68bc2826/*

McDonald, D., Papadopoulos, R., Benningfield. (25 de mayo de 2024). Reducción de la alucinación en LLM mediante la destilación del conocimiento: Un estudio de caso con Mistral Large y MMLU Benchmark. *TechRxiv*. DOI: 10.36227/techrxiv.171665607.76504195/v1

McMillen, E., & Himsl, F. (2011). *The Binding of Isaac* [Videojuego]. Edmund McMillen.

Microsoft. (2025). Descripción de los tokens. *Learn Microsoft* https://learn.microsoft.com/es-es/dotnet/ai/conceptual/understanding-tokens

Moreno, A. y Venegas, A. (2020). El videojuego como espejo de la sociedad contemporánea. *Barataria Revista Castellano-Manchega de Ciencias Sociales Nº 29*, pp. 1-8.https://doi.org/10.20932/barataria.v0i29.577

Pereira de Araujo, R., y Tiradentes, V. (2017). *Game worlds and creativity: The challenges of procedural content generation*. En A. Marcus & W. Wang (Eds.), Design, user experience, and usability: Designing pleasurable experiences (Lecture Notes in Computer Science, Vol. 10289, pp. 443–455). Springer.

Sandonnini, P. (05 de octubre de 2023). *LLaMA 2: Qué es y cómo utilizar el chatbot de código abierto de Meta.* Innovación Digital 360 LATAM. https://www.innovaciondigital360.com/i-a/llama-2-que-es-y-como-utilizar-el-chatbot-de-codigo-abierto-de-meta/

Servet, E. (2014). Diseño de entornos virtuales y narración de historias en videojuegos. *Metaverse Creativity 4(*1), pp. 75-91. https://doi.org/10.1386/mvcr.4.1.75_1

Shen, Z. (26 de agosto de 2022). Conferencia Web de SHS 144. Procedural Generation in Games: Focusing on Dungeons. London, Reino Unido. https://doi.org/10.1051/shsconf/202214402005

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Canton Ferrer, C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Xiang, J., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., & Scialom, T. (2023). *Llama 2: Open foundation and fine-tuned chat models*. arXiv. https://arxiv.org/abs/2307.09288

Valenzuela, A. (29 de julio de 2024). *Cuantificación para grandes modelos lingüísticos (LLM): Reduce eficazmente el tamaño de los modelos de IA.* Datacamp. https://www.datacamp.com/es/tutorial/quantization-for-large-language-models

Yu, D. 2012. *Spelunky*. Mossmouth. [Videojuego] Mossmouth.

Zhang, Y.,. Zhang G y Huang, X. A Survey of Procedural Content Generation for Games. (2022). *International Conference on Culture-Oriented Science and Technology (CoST)*, Lanzhou, China, 2022, pp. 186-190, doi: 10.1109/CoST57098.2022.00046.

Zia, T. (06 de marzo de 2024). Presentación de Gemma: el salto de código abierto de Google hacia la IA generativa. *Unite AI.* https://www.unite.ai/es/unveiling-gemma-googles-open-source-leap-into-generative-ai/