Inteligencia Artificial

Estado del Arte: Generación de Mazmorras

Benjamín Palma

24 de noviembre de 2024

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

Los juegos han acompañado a la humanidad desde sus inicios como herramientas de socialización, entretenimiento y, más importante aún, como catalizadores de creatividad y tecnología. Podemos observar cómo la industria de los juegos y la tecnología avanzan a pasos agigantados, reflejado en la creciente inversión y ganancias en este sector, así como en los avances en áreas como la computación cuántica, la robótica, la inteligencia artificial y el desarrollo de algoritmos. Técnicas como la generación procedural de mazmorras, fundamentales en el diseño de videojuegos, permiten la creación de entornos dinámicos, personalizables y únicos en cada partida. Al reducir la carga de diseño manual, estos algoritmos permiten a los desarrolladores centrarse en otros aspectos del juego, como la narrativa y la mecánica, fomentando así la innovación de nuevos conocimientos y tecnologías.

1. Introducción

La generación procedural se remonta a los años 70, con la llegada de los libros de *Dungeons & Dragons*, cuyo objetivo era recrear una aventura épica mediante una narrativa interactiva. En este contexto, la imaginación humana permite crear escenarios prácticamente infinitos y con interacciones complejas. Esta característica se hizo tan popular que fue adoptada en los videojuegos mediante el género "*Roguelike*", en honor al juego "Rogue" [3], el cual usa esta técnica en la creación de niveles tipo mazmorras, generando niveles únicos en cada partida. Hoy en día, los algoritmos de generación procedural buscan ofrecer esta misma libertad que caracteriza a *Dungeons & Dragons*, con altos grados de personalización, exploración, interacción y unicidad.

Siguiendo la línea de investigación en generación procedural, definida por Togelius et al. [6] como la creación de contenido por software junto a humanos o de manera autónoma, este proyecto se enfoca en explorar las estrategias óptimas para generar mazmorras, basándose en los modelos de contenido procedural categorizados por Breno et al. [9] La dificultad radica en generar un niveles coherente con barreras y llaves (las llaves abren las barreras), esto genera combinaciones no solucionables para el jugador, por lo que es una restriccion que se debe tomar en cuetna.

En las secciones posteriores, se introduce el trasfondo y la definición del problema; luego, en el Estado del Arte, se exploran las técnicas actuales y los modelos matemáticos en el contexto de la generación procedural; finalmente, se presentan las conclusiones basadas en el análisis de estrategias y su impacto en la generación de mazmorras.

2. Definición del Problema

De acuerdo con Togelius et. al [6] la generación procedural se refiere a algún software que pueden crear contenido en conjunto con humanos, diseñadores o autonómicamente. Van Der Linden [8] describe a los niveles de mazmorras (dungeon levels) como laberintos que integran de manera interrelacionada recompensas, desafíos y acertijos, los cuales el jugador debe superar para progresar en el juego.

El problema a tratar en este estudio involucra la creación de niveles de mazmorras con varios componentes clave: barreras B, llaves L, un mínimo de llaves requeridas L_R, habitaciones H y el coeficiente lineal C_L, estos parámetros son dados por el usuario. Las restricciones impuestas en el diseño de la mazmorra exigen que todas las habitaciones H estén conectadas de manera accesible desde la habitación inicial y que cada habitación tenga un máximo de cuatro conexiones adyacentes (arriba, abajo, izquierda y derecha). En estas habitaciones pueden colocarse puertas bloqueadas B que requieran llaves L para abrirse; cada llave abre una puerta específica, aunque algunas puertas pueden requerir varias llaves, y ciertas llaves pueden estar duplicadas. Para asegurar la resolubilidad del nivel, se define un mínimo de llaves requeridas L_R, y se diseña el algoritmo de modo que los valores de B, L, y L_R aseguren accesibilidad sin bloqueos insalvables.

Para medir la linealidad o ramificaci'on de la mazmorra, se introduce el coeficiente lineal C_L usado en [1], el cual mide el número promedio de habitaciones adyacentes, sin contar la habitación por la cual uno entra influyendo en el grado de ramificaci'on y controlando la linealidad del diseño, con valores entre 1 y 3. Este coeficiente se calcula promediando los valores individuales de cada habitación, y, por simplicidad, no se consideran casos con bucles.

El objetivo entonces es minimizar la diferencia entre los parámetros dados por el usuario sobre los valores de H, B, L, L_R y C_L los generados por el algoritmo, para que el diseño final de la mazmorra respete los par'ametros y criterios especificados en la mayor medida posible.

3. Estado del Arte

El diseño de entornos coherentes y visualmente atractivos es fundamental en la generación de mazmorras. Breno et al. [9] que proponen una taxonomía para la generación procedural de mazmorras, clasificándolas en cuatro categorías: tipo de mazmorra, elementos y mecánicas de juego, generación de contenido procedimental y estrategias de solución. Según la taxonomía los autores concluyen que creación de mazmorras condicionadas por barreras y llaves es un área subestudiada, mostrando 4 de 26 documentos revisados entre 2011 y 2019 en CM Digital Library, IEEE Xplore y Scopus abordaron este tema específico. En cuanto a las estrategias de solución, los algoritmos genéticos y constructivos son los más comunes para la generación de mazmorras. Sin embargo, en el contexto específico de problemas con barreras, no se ven estos métodos, sino enfoques alternativos, como gramática generativa, programación genética y programación de conjuntos de respuestas (*Answer Set Programming* ASP). Estudios recientes (2021-2024) [1, 5]

han investigado algoritmos evolutivos, también se aprecia un marco de trabajo de dos iteraciones en [1, 2], donde se basan en generar primero la mazmorra y luego la lógica de llaves y barreras. En el estado del arte actual (2023-2024), se han incorporado tecnologías como Modelos Largos de Lenguaje (LLM) para resolver problema, inspirados fuera de nuestro contexto en [7] donde usan a los modelos como interfaces para transformar problemas de lenguaje natural en código para solvers de satisfacción y optimización (ejemplo, MiniZinc). Y luego paper como [4] donde usan a estos modelos como núcleo para crear niveles con restricciones con marcos de trabajo iterativos, donde la generación, reparación y evaluación van convergiendo a una solución.

Considerando la experiencia previa en la aplicación de gramática generativa en este contexto, se presenta una oportunidad prometedora para explorar el potencial de los Modelos Largos de Lenguaje (LLM) en la resolución de nuestro problema. Nuestro enfoque innovador consiste en diseñar un modelo representable en texto, que pueda ser procesado mediante un gran modelo del lenguaje. Esta aproximación permite aprovechar las capacidades de comprensión y generación de lenguaje natural de los LLM. En este punto realizaremos dos pasos como mencionan en [1, 2], primero generaremos la mazmorra y validaremos su correcta generación, en el segundo paso ubicaremos las llaves y barreras. Posteriormente, evaluaremos la efectividad de la solución generado mediante algoritmos especializados basados en AStar que verifiquen el cumplimiento de las restricciones y requisitos específicos de nuestro problema. Este enfoque integrado combina la potencia de los LLM con la precisión de los algoritmos de evaluación, lo que nos permite abordar el problema desde una perspectiva más holística y eficiente.

Nuestro método se puede describir en las siguientes etapas:

1. Diseño del modelo representable en texto.
2. Generación de una solución de esquema de mazmorra mediante un gran modelo del lenguaje (LLM).
3. Evaluación de la solución mediante algoritmos de verificación de restricciones.
4. Instanciar llaves y barreras dentro de la Dungeon
5. Validar correcta instanciacion
6. Solución, sino iterar hasta 5 veces desde 4 en adelante con nueva información (heurísticas), sino volver a 2 y cambiar el esquema de la mazmorra.

4. Modelo Matemático

Modelamiento como problema de restricciones y optimización clásico (COP)

Función Objetivo

La función objetivo busca minimizar la diferencia absoluta entre los valores ideales y los alcanzados de habitaciones, llaves, barreras, linealidad y llaves necesarias:

$$f(x) = 2(\Delta_H + \Delta_L + \Delta_B + \Delta_{C_L}) + \Delta_{L_R}$$

donde:

- Δ_H es la diferencia absoluta en el número de habitaciones pedidas ($\widehat{H}$) y instanciadas (H).
- Δ_L es la diferencia en el número de llaves pedidas ($\widehat{L}$) y instanciadas (L).
- Δ_B es la diferencia en el número de barreras pedidas ($\widehat{B}$) y instanciadas (B).

- Δ_{C_L} es la diferencia en el coeficiente de linealidad pedidos ($\widehat{C_L}$) y instanciados (C_L).
- Δ_{L_R} es la diferencia en el número de llaves necesarias pedidas ($\widehat{L_R}$) y instanciadas (L_R).

En un inicio las instancias y lo pedido serán la misma, pero si no llegara a existir solución el algoritmo empezaría a variar primero las llaves necesarias pedidas y luego lo demás en orden de hacer el nivel soluciónale

Variables

- x_{ij}: Binaria $x_{ij} \in \{0, 1\}$, indica si existe una conexión entre la habitación i y la habitación j, donde $\{i \in \mathbb{N} \mid 0 \leq i \leq H,\ H \in \mathbb{N}\}$ y $\{j \in \mathbb{N} \mid 0 \leq j \leq H,\ H \in \mathbb{N} \wedge j \neq i\}$.
- k_b: Número de llaves requeridos para abrir la barrera $b \in \{b \in \mathbb{N} \mid 0 \leq b \leq B,\ B \in \mathbb{N}\}$, $k_b \in \mathbb{N}$.
- b_l: Barreras n-écima y su asignación a las llaves l-ésima (uno a uno). $b_l \in \{b_l \in \mathbb{N} \mid 0 \leq b_l \leq B,\ B \in \mathbb{N}\}$ y $l \in \{l \in \mathbb{N} \mid 0 \leq l \leq L,\ L \in \mathbb{N}\}$
- C_L: Coeficiente de linealidad (medido en función de la linealidad de la secuencia de conexión entre habitaciones). $C_L \in [1, 3]$
- L_R: Número de llaves necesarias para completar el nivel. $L_R \in \{L_R \in \mathbb{N} \mid B \leq L_R \leq L,\ B, L \in \mathbb{N}\}$

Restricciones

1. **Conectividad entre habitaciones**:
   - **Conexión mínima**: Todas las habitaciones deben estar conectadas al menos a otra habitación:
   $$\sum_j x_{ij} \geq 1, \quad \forall i$$
   - **Conexión máxima**: Cada habitación puede tener un máximo de 4 conexiones (arriba, derecha, abajo, izquierda):
   $$\sum_j x_{ij} \leq 4, \quad \forall i$$
2. **Asignación de llaves y barreras**:
   - Cada llave tiene una barrera asignada, pero una barrera puede requerir varias llaves:
   $$k_b \geq 1, \quad \forall b$$
3. **Compleción del nivel**:
   - El número de llaves asignadas debe ser suficiente para pasar todas las barreras y llegar al final del nivel:
   $$\sum_b k_b \leq l_N$$

Solución orientado a un algoritmo de 2 pasos evolutivo

Nuestra propuesta actual para la generación de mazmorras se basa en una representación mediante una estructura de árbol, como se describe en [1]. En esta representación, cada nodo del árbol corresponde a una habitación, y puede tener un máximo de tres hijos, que representan las habitaciones adyacentes hacia el oeste, sur y este, respectivamente. El nodo padre, por su parte, se sitúa siempre al norte del nodo actual. Este diseño permite una orientación espacial flexible de las habitaciones, evitando configuraciones rígidas y uniformes que limiten la variedad de mazmorras generadas. Como resultado, la mazmorra puede crecer homogéneamente en todas las direcciones, adaptándose mejor a los objetivos establecidos.

La representación en forma de árbol ofrece una flexibilidad estructural que resulta particularmente útil para aplicar operadores evolutivos como mutaciones y cruces. Por ejemplo, las mutaciones pueden consistir en la adición o eliminación de hojas en el árbol, lo que equivale a agregar o quitar habitaciones en la mazmorra. Esta capacidad de modificar dinámicamente la estructura permite ajustar el diseño de la mazmorra para que cumpla con requisitos específicos, como el número de habitaciones o el coeficiente lineal, que es un elemento clave en la evaluación de su diseño. El coeficiente lineal se calcula como el promedio de hijos de los nodo que no sean hojas.

Se inicializa la población inicial mediante una distribución normal para generar candidatos rápidamente aceptables pero diversos, unicamente enfocado en la cantidad de habitaciones, no se toma en cuenta el coeficiente lineal, ya que con las iteraciones ira convergiendo.

El proceso de cruzamiento entre mazmorras se pensaba realizar seleccionando subárboles de cada progenitor, contando con un parámetro que es max_depth que permite calcular la profundidad de un sub árbol contando desde la hoja hacia arriba, para posteriormente realizar un intercambio entre ellos. Este mecanismo genera dos nuevos individuos, o hijos, que combinan características estructurales de ambos padres. El objetivo principal del cruzamiento es introducir diversidad en la población de mazmorras, promoviendo diseños novedosos que se acerquen a los parámetros deseados, como la cantidad de habitaciones y el coeficiente lineal ideal. El gran problema es que por tiempo no se logro terminar la implementación, por lo que optamos por grandes tazas mutaciones (y variables)

Una vez generado un candidato de mazmorra que cumple con los criterios básicos, se procede a su refinamiento mediante la colocación aleatoria de barreras y llaves dentro de las habitaciones. Este paso inicial establece la distribución de elementos interactivos en el entorno. A continuación, se evalúa la viabilidad del diseño mediante un recorrido desde la raíz del árbol, utilizando un algoritmo Dijkstra para determinar si es posible recolectar las llaves necesarias para abrir cada barrera en el camino. Si el algoritmo confirma que todas las barreras pueden ser desbloqueadas, se establece la salida de la mazmorra justo detrás de la última barrera, garantizando que el diseño final sea tanto funcional como desafiante para el jugador.

Esta estrategia no solo asegura que las mazmorras generadas sean solucionables, sino que también permite iterar y ajustar el diseño para cumplir con parámetros específicos de dificultad, complejidad y distribución espacial. La combinación de una representación en forma de árbol con operadores evolutivos y técnicas de evaluación basadas en algoritmos de búsqueda asegura un equilibrio entre la aleatoriedad inherente al proceso y la capacidad de cumplir con los objetivos de diseño establecidos.

Instancias de Arboles Generados

parametros extra:

- steps = 30; pasos evolutivos.
- popSize = 50; Tamaño de la población.

- stdDev = 2.0; Desviación estándar base aplicada (crece proporcional a la instancia, ya que es proporcional a las habitaciones).
- minMut = 5; mínimo de mutaciones.
- maxMut = 15; máximo de mutaciones.
- crossoverProb = 0; Probabilidad de crossover, no se logro implementar correctamente.
- mutationProb = 1; Probabilidad de mutación, se aumento para remplazar la falta de crossover
- searchTry = 1000; iteraciones por árbol para buscar una solución mediante Dijkstra.

Instancia	**Nodos**	**Coef. Real**	**Coef. Objetivo**	**Evaluación**
Instancia 1	15	2.00	2.00	0
Instancia 2	20	1.19	1.00	0.1875
Instancia 3	20	1.73	2.00	0.2727
Instancia 4	25	1.14	1.00	0.1429
Instancia 5	30	2.00	2.00	0
Instancia 6	30	1.53	1.50	0.0263
Instancia 7	102	1.51	1.50	0
Instancia 8	500	1.50	1.50	0

Cuadro 1: Resultados de la evaluación de instancias de mazmorras generadas.

Análisis

- **Instancia 1**: 15/15 nodos, coeficiente 2.00, que es el valor ideal. El árbol es completamente resolvible, con una evaluación de 0, lo que indica una perfecta coincidencia con los objetivos.
- **Instancia 2**: 20/20 nodos, coeficiente 1.19, con una ligera discrepancia respecto al coeficiente objetivo de 1.00. La evaluación fue 0.1875, indicando una buena aproximación al objetivo.
- **Instancia 3**: 20/20 nodos, coeficiente 1.73. Similar a la Instancia 2, con una evaluación de 0.2727, lo que sugiere que el algoritmo ha logrado generar una mazmorra razonablemente cercana al coeficiente objetivo de 2.00.
- **Instancia 4**: 25/25 nodos, coeficiente 1.14, con una evaluación de 0.1429. El coeficiente real está algo lejos del objetivo de 1.00, pero aún así, la mazmorra es resolvible.
- **Instancia 5**: 30/30 nodos, coeficiente 2.00, lo que coincide exactamente con el coeficiente objetivo de 2.00, con una evaluación de 0, lo que indica una buena resolución del problema.
- **Instancia 6**: 30/30 nodos, coeficiente 1.53, con una evaluación de 0.0263. La evaluación muestra una ligera desviación respecto al objetivo, pero la mazmorra sigue siendo resolvible.
- **Instancia 7**: 100/100 nodos, coeficiente 1.51, con un coeficiente objetivo de 1.50, alcanzando una evaluación de 0, indicando que la mazmorra es completamente resolvible.
- **Instancia 8**: 500/- nodos, coeficiente 1.50, con un coeficiente objetivo de 1.50, pero no se logró resolver esta instancia en particular, es decir la segunda fase de barreras y llaves.

De las 8 instancias generadas, 7 fueron resueltas exitosamente, con un desempeño cercano o alcanzando los coeficientes objetivo. La instancia 8, sin embargo, no logró ser resuelta debido a la complejidad y al alto número de nodos requeridos. Esto se debe, en parte, a que la lógica de posicionamiento de barreras y llaves es aleatoria, por lo que, a mayor número de nodos, mayor entropía y, por ende, más estados no solucionables.

5. Conclusiones

La propuesta presentada combina una representación de mazmorras mediante árboles con operadores evolutivos y una evaluación basada en algoritmos de búsqueda, logrando diseños flexibles y funcionales. La estructura arbórea facilita la generación y modificación de mazmorras, permitiendo ajustes dinámicos para cumplir con requisitos específicos, como el número de habitaciones y el coeficiente lineal, que asegura equilibrio entre jugabilidad y complejidad.

En las pruebas realizadas, 7 de las 8 instancias generadas fueron completamente resolvibles, alcanzando un desempeño cercano o coincidente con los coeficientes objetivo. La instancia 8, con un mayor número de nodos, no fue resoluble debido a la complejidad añadida. Como posibles mejoras, se podría evaluar la implementación de una heurística para la colocación de barreras y llaves, además de sustituir Dijkstra por el algoritmo A* para resolver las instancias de manera más eficiente. Finalmente, se sugiere optimizar la función de crossover para permitir una mayor exploración del espacio de soluciones.

6. Bibliografía

Referencias

[1] Felipe Dumont and María-Cristina Riff. 2-step evolutionary algorithm for the generation of dungeons with lock door missions using horizontal symmetry. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1210–1218, 2024.

[2] Michael Cerny Green, Ahmed Khalifa, Athoug Alsoughayer, Divyesh Surana, Antonios Liapis, and Julian Togelius. Two-step constructive approaches for dungeon generation. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Tom Hatfield. Rise of the roguelikes: A genre evolves, 2013.

[4] Muhammad U. Nasir, Steven James, and Julian Togelius. Word2world: Generating stories and worlds through large language models, 2024.

[5] Leonardo Tortoro Pereira, Paulo Victor de Souza Prado, Rafael Miranda Lopes, and Claudio Fabiano Motta Toledo. Procedural generation of dungeonsâ maps and locked-door missions through an evolutionary algorithm validated with players. *Expert Systems with Applications*, 180:115009, 2021.

[6] Julian Togelius, Noor Shaker, and Mark J Nelson. Procedural content generation in games: A textbook and an overview of current research. *Togelius N. Shaker M. Nelson Berlin: Springer*, 2014.

[7] Dimos Tsouros, HÃ©lÃ¨ne Verhaeghe, Serdar KadÄ±oÄlu, *andTiasGuns*. *Holygrail*2,0 : *Fromnaturallanguagetoconstraintmodels*, 2023.

[8] Roland Van Der Linden, Ricardo Lopes, and Rafael Bidarra. Procedural generation of dungeons. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(1):78–89, 2013.

[9] Breno M. F. Viana and Selan R. dos Santos. A survey of procedural dungeon generation. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 29–38, 2019.