



UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUSTAVO GURGEL MEDEIROS

**GERAÇÃO DE MAPAS PARA JOGOS ROGUELIKE A PARTIR DE DESCRIÇÕES
TEXTUAIS UTILIZANDO MODELOS DE LINGUAGEM DE LARGA ESCALA**

QUIXADÁ

2026

GUSTAVO GURGEL MEDEIROS

GERAÇÃO DE MAPAS PARA JOGOS ROGUELIKE A PARTIR DE DESCRIÇÕES
TEXTUAIS UTILIZANDO MODELOS DE LINGUAGEM DE LARGA ESCALA

Trabalho de Conclusão de Curso apresentado ao
Curso de Graduação em Ciência da Computação
do Campus Quixadá da Universidade Federal
do Ceará, como requisito parcial à obtenção do
grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Cristiano Bacelar
de Oliveira.

QUIXADÁ

2026

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

- M439g Medeiros, Gustavo Gurgel.
 Geração de Mapas para Jogos Roguelike a Partir de Descrições Textuais Utilizando Modelos de
 Linguagem de Larga Escala / Gustavo Gurgel Medeiros. – 2026.
 59 f. : il. color.
- Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá,
 Curso de Ciência da Computação, Quixadá, 2026.
 Orientação: Prof. Dr. Cristiano Bacelar de Oliveira.
1. Geração Procedural de Conteúdo. 2. Modelos de Linguagem de Larga Escala. 3. Jogos Roguelike. I.
 Título.

CDD 004

GUSTAVO GURGEL MEDEIROS

GERAÇÃO DE MAPAS PARA JOGOS ROGUELIKE A PARTIR DE DESCRIÇÕES
TEXTUAIS UTILIZANDO MODELOS DE LINGUAGEM DE LARGA ESCALA

Trabalho de Conclusão de Curso apresentado ao
Curso de Graduação em Ciência da Computação
do Campus Quixadá da Universidade Federal
do Ceará, como requisito parcial à obtenção do
grau de bacharel em Ciência da Computação.

Aprovada em: 21/01/2026.

BANCA EXAMINADORA

Prof. Dr. Cristiano Bacelar de Oliveira (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Paulo de Tarso Guerra
Universidade Federal do Ceará (UFC)

Prof. Dr. João Vilnei de Oliveira Filho
Universidade Federal do Ceará (UFC)

Dedico este trabalho a Deus, pela sabedoria concedida; à minha família, base de tudo; aos amigos, que tornaram a caminhada mais leve; e aos meus mestres, pelos ensinamentos que levarei para a vida.

AGRADECIMENTOS

A Deus, primeiramente, pelo dom da vida, pela sabedoria e por me conceder forças para superar os obstáculos e chegar até aqui.

Aos meus pais, Caroline Maria e Paulo Fernando, meu porto seguro, pelo amor incondicional, pelos sacrifícios feitos em prol da minha educação e por sempre acreditarem em mim.

À minha irmã, Lara Maria, pelo companheirismo e carinho.

Aos meus amigos, que fizeram essa caminhada mais leve e alegre. Um agradecimento especial ao meu amigo Mário Martins, pela parceria e presença constante ao longo desta jornada.

Ao meu orientador, Prof. Dr. Cristiano Bacelar, pela confiança e condução deste trabalho.

Aos professores membros da banca examinadora, Prof. Dr. Paulo de Tarso e Prof. Dr. João Vilnei, pelo tempo dedicado e pelas contribuições para o aprimoramento desta pesquisa.

Aos meus professores, por todos os ensinamentos compartilhados em sala de aula, que foram essenciais para a minha formação profissional e humana.

"Quem observa o vento, nunca semeará, e o que
olha para as nuvens nunca segará." (BÍBLIA
SAGRADA, Eclesiastes 11:4.)

RESUMO

O gênero de jogos *roguelike*, conhecido pela alta rejogabilidade proporcionada pela *Procedural Content Generation* (PCG), frequentemente apresenta desafios no que tange ao controle fino e à criação de conteúdo com intenções específicas por parte dos desenvolvedores. A manipulação de parâmetros complexos e pouco flexíveis em algoritmos de PCG tradicionais torna o processo de design pouco intuitivo. Este trabalho explora uma abordagem para superar essa limitação, propondo um sistema que utiliza *Large Language Model* (LLM) para gerar mapas a partir de descrições textuais em linguagem natural. Reconhecendo as limitações inerentes das LLMs em raciocínio espacial e geométrico, a pesquisa desenvolveu uma arquitetura híbrida que desacopla responsabilidades: algoritmos determinísticos tradicionais gerenciam a topologia estrutural e a conectividade da masmorra, enquanto a LLM atua como um "agente criativo" responsável pela geração semântica de narrativas, inimigos e itens. Para materializar esses elementos, foi implementado um *pipeline* que integra *Structured Outputs* e a técnica de *Retrieval-Augmented Generation* (RAG) sobre um banco vetorial curado de *tiles*, garantindo a consistência visual e mitigando alucinações. Um jogo funcional foi desenvolvido para interpretar os ativos gerados e renderizar níveis jogáveis. A validação, realizada através de métricas de Reconstrução Semântica e Coerência sobre três modelos distintos, demonstrou que modelos menores executados localmente apresentam desempenho comparável a grandes modelos. Os resultados confirmam a viabilidade técnica de associar linguagem natural a ativos de jogo concretos, estabelecendo uma base sólida para sistemas de criação "*text-to-map*" que preservam a intenção do usuário.

Palavras-chave: geração procedural de conteúdo; modelos de linguagem de larga escala; jogos roguelike; geração de mapas.

ABSTRACT

The roguelike game genre, known for its high replayability provided by PCG, often presents challenges regarding fine-grained control and the creation of content with specific developer intentions. Manipulating complex and rigid parameters in traditional PCG algorithms renders the design process unintuitive. This work explores an approach to overcome this limitation by proposing a system that uses LLMs to generate maps from natural language textual descriptions. Recognizing the inherent limitations of LLMs in spatial and geometric reasoning, the research developed a hybrid architecture that decouples responsibilities: traditional deterministic algorithms manage the structural topology and dungeon connectivity, while the LLM acts as a "creative agent" responsible for the semantic generation of narratives, enemies, and items. To materialize these elements, a pipeline was implemented integrating Structured Outputs and the RAG technique over a curated vector database of tiles, ensuring visual consistency and mitigating hallucinations. A functional game prototype was developed to interpret the generated assets and render playable levels. Validation, performed through Semantic Reconstruction and Coherence metrics across three distinct models, demonstrated that smaller models running locally offer performance comparable to large models. The results confirm the technical feasibility of associating natural language with concrete game assets, establishing a solid foundation for "text-to-map" creation systems that preserve user intention.

Keywords: procedural content generation; large language models; roguelike games; map generation.

LISTA DE FIGURAS

Figura 1	– Screenshot do jogo <i>Rogue</i>	17
Figura 2	– <i>Dead Cells</i> (2018)	18
Figura 3	– Mundos do <i>Minecraft</i> como estes são todos gerados algoritmicamente.	21
Figura 4	– Exemplo de funções de densidade que utilizam ruído 3D para geração de biomas e altura de terreno no <i>Minecraft</i>	22
Figura 5	– Exemplo de mapa gerado utilizando gramáticas de grafos	23
Figura 6	– Arquitetura do sistema <i>MarioGPT</i> . Um prompt textual é processado por um encoder de texto, cujas saídas são usadas em um mecanismo de atenção cruzada com as camadas <i>Generative Pre-trained Transformer</i> (GPT). As camadas GPT também recebem um nível "tokenizado" inicial. O sistema então prediz a continuação do nível "tokenizado", que é convertido para um nível de jogo visual.	26
Figura 7	– Fluxograma das etapas metodológicas do trabalho.	29
Figura 8	– Exemplo de falha na geração direta de topologia por LLM. Observam-se inconsistências estruturais graves, destacadas como <i>tiles</i> ausentes no chão da sala e vazamento de estruturas para fora dos limites lógicos do mapa.	30
Figura 9	– Arquitetura Híbrida proposta: O fluxo inicia com a descrição do tema, que alimenta tanto o algoritmo de PCG quanto a LLM. A LLM gera assets semânticos (inimigos, itens e tiles) utilizando um Vector Store de texturas (RAG), enquanto o algoritmo de PCG utiliza esses assets para construir a estrutura espacial da masmorra.	31
Figura 10	– Fluxo de geração do Pacote de Assets: da descrição inicial à instanciação dos objetos via RAG.	32
Figura 11	– Tilemap completo do pacote " <i>1-Bit Pack</i> " do site <i>Kenney</i> , contendo a totalidade dos ativos gráficos disponíveis antes da filtragem.	33
Figura 12	– Exemplos de execução da função de busca vetorial para três categorias distintas: Items ("A great sword"), Entities ("A dwarf") e Environments ("A stone wall").	34
Figura 13	– Fluxo de geração de um asset estruturado: a descrição textual é processada, combinada com a textura recuperada do Vector Store e validada via Pydantic para criar um objeto de jogo funcional com atributos discretos.	36

Figura 14 – Visualização dos níveis, jogador e objetivo final gerados (GPT-120b).	44
Figura 15 – Armas geradas (GPT-120b).	44
Figura 16 – Inimigos gerados (GPT-120b).	45
Figura 17 – Visualização dos níveis, jogador e objetivo final gerados (Llama-17b).	45
Figura 18 – Armas geradas (Llama-17b).	46
Figura 19 – Inimigos gerados (Llama-17b).	46
Figura 20 – Visão geral da interface do jogo, destacando HUD, logs e inventário.	48
Figura 21 – Sistema de Campo de Visão: distinção entre áreas visíveis, exploradas e não exploradas.	49
Figura 22 – Três exemplos de níveis gerados proceduralmente com diferentes configurações espaciais.	50

LISTA DE ABREVIATURAS E SIGLAS

ANN	<i>Artificial Neural Networks</i>
ASCII	<i>American Standard Code for Information Interchange</i>
BERT	<i>Bidirectional Encoder Representations from Transformers)</i>
CNN	<i>Convolutional Neural Networks</i>
FOV	<i>Field of View</i>
GEDC	<i>Game Environment Design Creator</i>
GPT	<i>Generative Pre-trained Transformer</i>
GPU	<i>Graphics Processing Unit</i>
HUD	<i>Heads-Up Display</i>
JSON	<i>JavaScript Object Notation</i>
LLM	<i>Large Language Model</i>
LSTM	<i>Long Short-Term Memory</i>
MoE	<i>Mixture of Experts</i>
NLP	<i>Natural Language Processing</i>
NPC	<i>Non-Player Character</i>
PCG	<i>Procedural Content Generation</i>
RAG	<i>Retrieval-Augmented Generation</i>
REST	<i>Representational State Transfer</i>
RNN	<i>Recurrent Neural Network</i>
RPG	<i>Role-Playing Game</i>
TPU	<i>Tensor Processing Unit</i>

SUMÁRIO

1	INTRODUÇÃO	13
1.1	Contribuições	14
2	FUNDAMENTAÇÃO TEÓRICA	16
2.1	Roguelikes e Roguelites	16
2.2	PCG em Jogos	19
2.2.1	<i>Definição e Objetivos da PCG</i>	<i>19</i>
2.2.2	<i>Aplicações da PCG em Jogos</i>	<i>20</i>
2.2.3	<i>Vantagens e Desafios da PCG</i>	<i>20</i>
2.2.4	<i>Técnicas Comuns de PCG</i>	<i>21</i>
2.3	LLMs	23
2.3.1	<i>Definição e Evolução Histórica dos LLMs</i>	<i>24</i>
2.3.2	<i>Transformers e Mecanismo de Atenção</i>	<i>24</i>
2.3.3	<i>Treinamento e Capacidades dos LLMs</i>	<i>25</i>
2.3.4	<i>LLMs Aplicados à PCG</i>	<i>25</i>
3	TRABALHOS RELACIONADOS	27
4	METODOLOGIA	29
4.1	Visão Geral da Metodologia	29
4.2	Definição da Arquitetura do Gerador de Mapas	29
4.2.1	<i>Geração de Assets via LLM</i>	<i>30</i>
4.2.2	<i>Construção Estrutural via PCG Tradicional</i>	<i>31</i>
4.3	Implementação do Gerador de Assets	32
4.3.1	<i>Dataset e Pré-processamento Visual</i>	<i>33</i>
4.3.2	<i>Stack Tecnológico e Busca Vetorial</i>	<i>34</i>
4.3.3	<i>Geração de Entidades com Saídas Estruturadas</i>	<i>35</i>
4.3.4	<i>Abstração de Design e Valores Discretos</i>	<i>35</i>
4.3.5	<i>Interface de Comunicação</i>	<i>35</i>
4.4	Implementação do Gerador de Mapas	36
4.4.1	<i>Escolha Tecnológica e Arquitetura Base</i>	<i>36</i>
4.4.2	<i>Integração de Dados e Desserialização</i>	<i>37</i>
4.4.3	<i>Algoritmo de Geração Estrutural</i>	<i>37</i>

4.4.4	<i>Povoamento e Progressão de Dificuldade</i>	38
4.4.5	<i>Ciclo de Gameplay e Condição de Vitória</i>	38
4.5	Definição das Métricas de Validação	39
4.5.1	<i>Métrica de Coerência (Coherence)</i>	39
4.5.2	<i>Métrica de Reconstrução Semântica</i>	40
5	RESULTADOS	41
5.1	Configuração Experimental	41
5.1.1	<i>Modelos e Cenários de Teste</i>	41
5.2	Análise Quantitativa	42
5.3	Análise Qualitativa	43
5.3.1	<i>Caso de Estudo 1: The Derelict Starship (GPT-120b)</i>	43
5.3.2	<i>Caso de Estudo 2: The Living Hive (Llama-17b)</i>	44
5.4	Jogo Roguelike	47
5.4.1	<i>Interface e Ciclo de Gameplay</i>	47
5.4.2	<i>Sistema de Campo de Visão (FOV)</i>	48
5.4.3	<i>Geração Procedural e Progressão de Dificuldade</i>	49
6	CONCLUSÕES E TRABALHOS FUTUROS	51
6.1	Limitações e Trabalhos Futuros	52
	REFERÊNCIAS	53
	APÊNDICE A – Exemplo de Asset Bundle em JSON	58

1 INTRODUÇÃO

Os jogos digitais representam uma área de significativa importância na sociedade contemporânea, com aplicações que abrangem desde o entretenimento até fins educacionais. A indústria de jogos eletrônicos tem demonstrado um crescimento notável, consolidando-se como um mercado de alcance global sem precedentes (Wijman, 2023).

Nesse contexto, diversos jogos do gênero *roguelike* têm alcançado grande sucesso, com títulos notáveis como *Hollow Knight*¹, *Undermine*² e *Dead Cells*³ destacando-se pela capacidade de proporcionar experiências desafiadoras e imersivas. O gênero *roguelike*, um subgênero de *Role-Playing Game* (RPG), caracteriza-se fundamentalmente pela exploração de masmorras geradas aleatoriamente, combate tipicamente baseado em turnos e a morte permanente do personagem, que impõe o recomeço do jogo após uma derrota (Laurindo *et al.*, 2024).

Um elemento fundamental que define a experiência em muitos jogos *roguelike* é a PCG. PCG é uma técnica que utiliza algoritmos para criar automaticamente conteúdos de jogo, como mapas e cenários, sendo fundamental para garantir que cada sessão de jogo seja única e ofereça alta rejogabilidade (Sepänmaa, 2024). Embora a PCG seja poderosa para gerar uma vasta quantidade de conteúdo, um de seus desafios históricos reside na dificuldade de controlar o resultado. Tradicionalmente, os desenvolvedores manipulam parâmetros complexos e abstratos para guiar o processo gerador, o que muitas vezes torna a criação de conteúdo com intenções específicas uma tarefa pouco intuitiva e iterativa (Lazaridis; Fragulis, 2024).

A busca por maior controle na PCG tem sido um campo ativo de pesquisa. Contudo, gerar conteúdo que reflita intenções específicas e restrições de alto nível permanece um desafio. Recentemente, os avanços em LLMs abriram novas e promissoras formas para abordar este problema. A capacidade inerente dos LLMs de compreender e processar linguagem natural oferece uma interface fundamentalmente nova para a geração de conteúdo: em vez de ajustar parâmetros, os desenvolvedores ou mesmo os jogadores poderiam simplesmente descrever o conteúdo desejado textualmente.

Essa abordagem, frequentemente denominada "*text-to-level*" ou "*text-to-map*", transforma o LLM em um tradutor de intenções humanas para estruturas de jogo concretas. Um

¹ Página da Steam do Hollow Knight (Acesso em: 23 jan. 2026): https://store.steampowered.com/app/367520/Hollow_Knight/

² Página da Steam do UnderMine (Acesso em: 23 jan. 2026): <https://store.steampowered.com/app/656350/UnderMine/>

³ Página da Steam do Dead Cells (Acesso em: 23 jan. 2026): https://store.steampowered.com/app/588650/Dead_Cells/

trabalho pioneiro nesta área, *MarioGPT*, demonstrou a viabilidade de treinar um LLM para gerar níveis de *Super Mario Bros* a partir de descrições simples como "muitos canos, poucos inimigos, elevação baixa" (Sudhakaran *et al.*, 2023). Este estudo provou que os LLMs podem não apenas gerar sequências textuais, mas também produzir conteúdo estruturado e funcional para jogos, abordando um dos principais desafios da PCG.

Inspirado por esses avanços, este trabalho investiga a aplicação dessa abordagem no domínio dos jogos *roguelike*. Esse gênero é caracterizado por mapas com restrições estruturais, como a conectividade entre salas, a disposição de corredores e a distribuição lógica de inimigos e tesouros. Sendo assim, o objetivo principal é investigar como os LLMs podem traduzir uma descrição textual em um mapa funcional, garantindo que a estrutura resultante se adéque a essas convenções.

Este estudo busca desenvolver um sistema que utilize um LLM para interpretar descrições textuais e produzir uma representação de mapa estruturada, que possa ser posteriormente renderizada por um motor de jogo. A pesquisa se concentrará na eficácia de técnicas de engenharia de *prompt* para guiar o modelo, garantindo que os mapas gerados não apenas correspondam à descrição, mas também sejam funcionalmente válidos, respeitando as restrições estruturais do gênero mencionadas anteriormente (como conectividade de salas e disposição de corredores). Acredita-se que tal abordagem oferece um método de criação de conteúdo procedural mais intuitivo e expressivo em comparação com técnicas algorítmicas tradicionais, como Perlin Noise (Perlin, 1985) e Random Walk (Pearson, 1905), contribuindo com conhecimento técnico e metodológico.

1.1 Contribuições

Considerando os resultados obtidos e a validação do sistema proposto, este trabalho apresenta contribuições técnicas e práticas para a área de PCG assistida por Inteligência Artificial. As principais contribuições podem ser sintetizadas em três pilares:

- **Arquitetura Híbrida de Geração "Text-to-Map":** O desenvolvimento de uma metodologia que desacopla a responsabilidade criativa da responsabilidade estrutural. Esta abordagem mitiga as limitações de raciocínio espacial inerentes aos LLMs, delegando a topologia a algoritmos determinísticos enquanto utiliza o modelo de linguagem para enriquecer semanticamente o jogo (narrativa, inimigos e itens). O sistema materializa essa criação através de arquivos estruturados em *JavaScript Object Notation* (JSON),

garantindo interoperabilidade com motores de jogo (um exemplo de um artefato JSON gerado encontra-se disponível no Apêndice A).

- **Dataset Curado para RAG em Jogos:** A implementação de um *pipeline* técnico que integra *Structured Outputs* e RAG. Neste contexto, a curadoria e anotação manual de um *dataset*, composto por 489 pares de descrições e imagens, constitui uma contribuição técnica relevante deste trabalho, estabelecendo uma base sólida para a associação entre linguagem natural e ativos de *pixel art*.
- **Ferramentas de Código Aberto:** A disponibilização de dois artefatos de *software* funcionais e integrados, que servem como prova de conceito e base para pesquisas futuras. Os códigos-fonte do projeto foram tornados públicos nos seguintes repositórios:
 - **Gerador de Assets (Python):** <https://github.com/GusGurgel/tcc-roguelike-assets-generator>
 - **Gerador de Mapas (Godot):** <https://github.com/GusGurgel/tcc-roguelike>

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados os principais conceitos fundamentais para o desenvolvimento deste trabalho. A Seção 2.1 trata dos aspectos históricos e mecânicas fundamentais dos gêneros *roguelike* e *roguelite*. Na Seção 2.2, discute-se a PCG, abordando sua definição, objetivos, aplicações, vantagens, desafios e técnicas comuns. Por fim, a Seção 2.3 oferece uma descrição sobre os LLMs, considerando sua definição, evolução, arquitetura, treinamento, capacidades e aplicação na PCG.

2.1 Roguelikes e Roguelites

O gênero *roguelike* compreende uma categoria de jogos de RPG eletrônicos caracterizados por uma série de mecânicas distintas, das quais a morte permanente e a geração procedural de níveis são as mais proeminentes. O termo deriva diretamente do jogo seminal *Rogue*^{1,2}, desenvolvido por Michael Toy, Glenn Wichman e Ken Arnold e lançado em 1980. *Rogue* surgiu em um contexto no qual jogos de aventura baseados em texto, como *Colossal Cave Adventure*, e RPGs de mesa, como *Dungeons & Dragons*, eram populares, mas a representação gráfica e a interatividade dinâmica em computadores pessoais ainda estavam em seus estágios iniciais (Harris, 2020).

Em *Rogue*, o jogador assume o papel de um aventureiro explorando uma masmorra perigosa com o objetivo de recuperar o Amuleto de *Yendor* e retornar à superfície. O jogo utilizava caracteres no formato *American Standard Code for Information Interchange* (ASCII) para representar graficamente o personagem do jogador (geralmente um '@'), monstros (representados por letras diversas), itens e o layout da masmorra, como ilustrado na Figura 1. Esta abordagem visual, embora primitiva para os padrões atuais, era inovadora por permitir uma representação espacial do ambiente do jogo, que era gerado aleatoriamente a cada nova partida, uma característica fundamental que definiria o gênero.

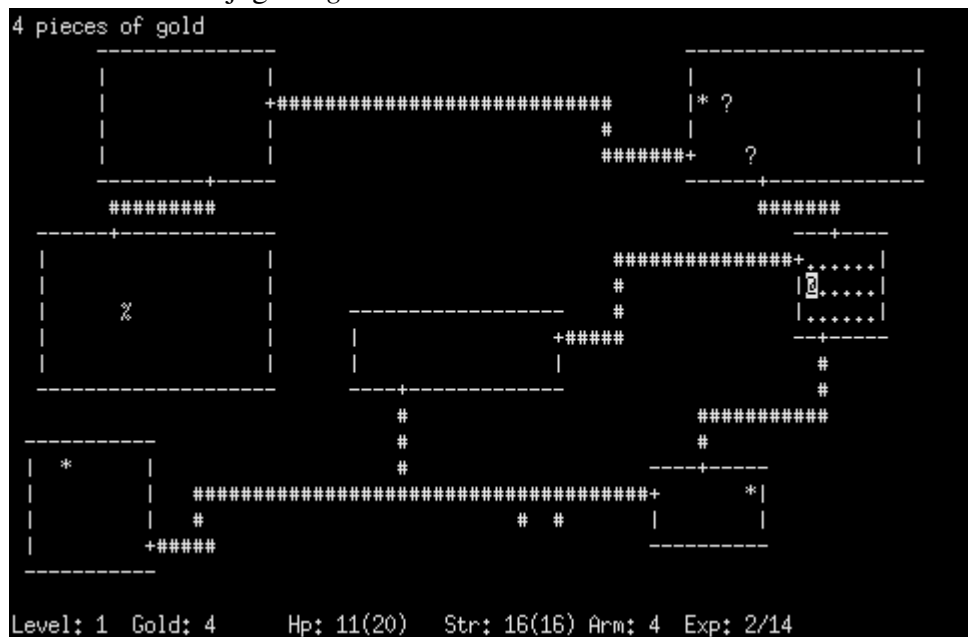
O impacto de *Rogue* foi significativo, estabelecendo as fundações para um gênero que persistiria e evoluiria por décadas (Harris, 2020). Desenvolvido para terminais *Unix* e utilizando a biblioteca *Curses*³ para manipulação de tela, *Rogue* introduziu avanços importantes

¹ Artigo sobre *Rogue* no site *RogueBasin*, uma das principais Wikis de *Roguelikes* tradicionais (Acesso em: 23 jan. 2026): <https://www.roguebasin.com/index.php/Rogue>

² Manual do jogo *Rogue* (Acesso em: 23 jan. 2026): <https://britzl.github.io/roguearchive/files/misc/EpyxRogueDOSManual/manual.htm>

³ *Curses* (Acesso em: 23 jan. 2026): <https://www.ibm.com/docs/en/aix/7.2.0?topic=concepts-curses-library>

Figura 1 – Screenshot do jogo *Rogue*



Fonte: Disponível em: [https://en.wikipedia.org/wiki/Rogue_\(video_game\)](https://en.wikipedia.org/wiki/Rogue_(video_game)). Acesso em: 23 jan. 2026.

não apenas para os RPGs de computador, mas para o gênero de jogos de PC em geral. A geração procedural, a trama básica de exploração e retorno, e a exibição baseada em texto tornaram-se características centrais e recorrentes nos jogos que se seguiram e que se autodenominariam *roguelikes* (Harris, 2020).

Um dos pilares fundamentais do gênero *roguelike* é a mecânica de morte permanente. Essa característica implica que, quando o personagem do jogador morre, todo o progresso naquela partida específica é perdido, e o jogador deve recomeçar do zero. A morte permanente intensifica a tensão e o valor de cada decisão tomada pelo jogador, pois um erro pode ter consequências fatais e irreversíveis para aquela sessão de jogo. Este ciclo de tentativa, falha e aprendizado é central para a experiência *roguelike*, forçando os jogadores a internalizar as mecânicas do jogo e a desenvolverem suas próprias habilidades e estratégias, em vez de dependerem apenas da progressão virtual do personagem.

A natureza punitiva da morte permanente é frequentemente vista como um elemento que torna as escolhas e os riscos mais significativos. Em muitos jogos de RPG tradicionais, a morte do personagem é uma inconveniência temporária, resolvida com o carregamento de um jogo salvo. Nos *roguelikes* clássicos, no entanto, a perda do personagem significa o fim daquela jornada específica, e o jogador leva consigo apenas o conhecimento adquirido para a próxima tentativa. Este foco na habilidade do jogador, em detrimento da acumulação de recursos ou níveis do personagem que persistem entre as mortes, é um diferencial marcante do gênero (Botta,

2011).

De forma complementar à morte permanente, a PCG é o outro pilar essencial dos *roguelikes*. Em *Rogue*, e em seus sucessores, os níveis da masmorra, a localização de inimigos, itens e tesouros são gerados algoritmicamente a cada nova partida. Isso garante uma alta rejogabilidade, pois cada sessão de jogo apresenta um desafio único e imprevisível. Mesmo os desenvolvedores de *Rogue* não poderiam prever a configuração exata de uma masmorra, o que contribuía para a sensação de exploração genuína (Short; Adams, 2017).

Por muitas décadas, o gênero *roguelike* permaneceu um nicho, apreciado por uma comunidade dedicada de fãs, mas foi nesta última década que ele emergiu para o *mainstream*, influenciando uma vasta gama de jogos independentes e até mesmo títulos de grande orçamento (Bycer, 2019). Com essa popularização, surgiu o termo *roguelite*⁴ para descrever jogos que incorporam elementos chave dos *roguelikes*, como a morte permanente e a geração procedural, mas que se desviam da fórmula clássica, muitas vezes introduzindo sistemas de meta-progressão. Essa meta-progressão permite que certos desbloqueios, habilidades ou recursos sejam carregados entre as partidas, suavizando a curva de dificuldade e oferecendo uma sensação de progresso contínuo mesmo após a falha (Albert, 2023).

Figura 2 – Dead Cells (2018)



Fonte: Disponível em: https://store.steampowered.com/app/588650/Dead_Cells/. Acesso em: 23 jan. 2026.

Os *roguelites* focam frequentemente em alcançar um final definido, em contraste com os *roguelikes* clássicos, nos quais o foco está mais na jornada e na superação de desafios

⁴ Debate sobre o significado do termo *Roguelite* (Acesso em: 23 jan. 2026): <https://www.gamedeveloper.com/design/the-roguelike-debate---roguelikes-vs-roguelites>

imediatos de cada partida. A persistência de elementos entre as tentativas, como melhorias de atributos base, desbloqueio de novas armas ou habilidades, mecânicas centrais em títulos como o visualmente impactante *Dead Cells* (Figura 2), torna os *roguelites* mais acessíveis a um público mais amplo, que pode se sentir frustrado pela natureza implacável da morte permanente total. Outros títulos como *Spelunky*⁵, *FTL: Faster Than Light*⁶, e *The Binding of Isaac*⁷ são também exemplos proeminentes que aplicaram a fórmula *roguelike* a diferentes gêneros, introduzindo variações que levaram à necessidade de uma nova categorização, embora o debate sobre as definições exatas de *roguelike* e *roguelite* continue sendo debatido entre fãs e desenvolvedores (Cartlidge, 2024).

2.2 PCG em Jogos

Esta seção visa apresentar uma visão geral sobre a PCG, abordando sua definição, objetivos, aplicações, vantagens, desafios e as técnicas mais comuns empregadas, com ênfase em sua relevância para a geração de níveis e narrativas, especialmente no contexto de jogos do gênero *roguelike*.

2.2.1 Definição e Objetivos da PCG

A PCG refere-se fundamentalmente à criação de conteúdo de jogo de forma algorítmica, em vez de manualmente. Segundo Togelius *et al.* (2011a), PCG é a geração de conteúdo de jogo por algoritmos, abrangendo desde elementos visuais até mecânicas de jogo. Smith *et al.* (2011) complementa esta visão, destacando que o objetivo da PCG não é apenas automatizar a criação, mas também possibilitar a emergência de novas formas de experiência de jogo, como a rejogabilidade infinita ou mundos de jogo vastos e únicos. O objetivo principal da PCG é, portanto, utilizar processos computacionais para produzir uma grande quantidade e variedade de conteúdo, muitas vezes com recursos de desenvolvimento limitados, ou para criar experiências que seriam impossíveis ou impraticáveis de serem desenvolvidas por métodos tradicionais de design manual. Para o escopo deste trabalho, a PCG é entendida como o conjunto de técnicas e algoritmos empregados para gerar, automaticamente ou com intervenção mínima do designer, elementos como níveis, narrativas e outros componentes de jogos.

⁵ Site oficial do *Spelunky* (Acesso em: 23 jan. 2026): <https://www.spelunkyworld.com/>

⁶ Site oficial do *FTL: Faster Than Light* (Acesso em: 23 jan. 2026): <https://subsetgames.com/ftl.html>

⁷ Página da Steam do *The Binding of Isaac* (Acesso em: 23 jan. 2026): https://store.steampowered.com/app/113200/The_Binding_of_Isaac/

2.2.2 Aplicações da PCG em Jogos

A PCG encontra um vasto campo de aplicação no desenvolvimento de jogos, permeando diversos aspectos da experiência do jogador. Dentre os tipos de conteúdo mais comumente gerados proceduralmente, destacam-se os mapas, que constituem um foco central deste trabalho, devido à sua importância para a exploração e desafio em jogos. Além da arquitetura de níveis, a PCG é amplamente utilizada para criar itens com atributos variados, posicionar inimigos e gerar *Non-Player Character* (NPC) com características distintas. A geração também se estende a elementos audiovisuais, como texturas, modelos 3D, música e efeitos sonoros, contribuindo para a maior diversidade dos mundos virtuais. Outra aplicação relevante é a criação de missões, que podem ser adaptadas dinamicamente ao progresso do jogador ou a eventos do jogo.

Diversos jogos comerciais e independentes utilizam PCG de forma proeminente para alcançar seus objetivos de design. Títulos como *Spelunky* e seus sucessores são exemplos de como a PCG pode criar níveis desafiadores e altamente jogáveis. *No Man's Sky*⁸ utiliza PCG para gerar um universo de planetas, cada um com sua flora, fauna e topografia únicos, oferecendo uma escala de exploração virtualmente infinita. *Minecraft*⁹ é outro exemplo icônico, no qual vastos mundos são gerados proceduralmente, permitindo que os jogadores explorem e modifiquem o terreno de maneiras criativas. O gênero *roguelike*, desde seus precursores como *Rogue* até títulos modernos, é intrinsecamente ligado à PCG, utilizando-a para gerar masmorras, itens e encontros, garantindo que cada partida seja uma nova experiência.

2.2.3 Vantagens e Desafios da PCG

A adoção da PCG no desenvolvimento de jogos traz consigo um conjunto de vantagens significativas, mas também impõe desafios consideráveis que precisam ser gerenciados pelos desenvolvedores. Entre as principais vantagens, figura a capacidade de oferecer alta jogabilidade (Lazaridis; Fragulis, 2024). A PCG permite a criação de mundos vastos e variados que seriam inviáveis de serem construídos manualmente, como observado em *No Man's Sky*. Outro benefício importante é a eficiência de armazenamento, já que o conteúdo é gerado em tempo de execução ou durante o desenvolvimento a partir de algoritmos compactos, em vez de armazenar grandes volumes de dados pré-fabricados. Adicionalmente, a PCG possui o potencial para adaptação dinâmica do conteúdo às ações ou ao perfil do jogador, criando experiências mais

⁸ Site oficial do *No Man's Sky* (Acesso em: 23 jan. 2026): <https://www.nomanssky.com/>

⁹ Site oficial do *Minecraft* (Acesso em: 23 jan. 2026): <https://www.minecraft.net/>

Figura 3 – Mundos do *Minecraft* como este são todos gerados algoritmicamente.



Fonte: Disponível em: https://minecraft.wiki/w/World_generation. Acesso em: 23 jan. 2026.

personalizadas.

Contudo, a PCG não está isenta de desafios. O controle de qualidade do conteúdo gerado pode ser complexo, pois é preciso garantir que os resultados sejam consistentemente bons, esteticamente agradáveis e livres de artefatos indesejados ou bugs (Lazaridis; Fragulis, 2024). A garantia de jogabilidade, incluindo a solvabilidade níveis ou quebra-cabeças e a "justiça" das situações de jogo, é outra preocupação central; um nível gerado proceduralmente deve ser sempre solucionável e oferecer um desafio adequado. A previsibilidade excessiva pode minar a sensação de descoberta, enquanto a aleatoriedade descontrolada pode gerar conteúdo incoerente ou frustrante. Finalmente, um dos desafios mais debatidos é a dificuldade em replicar o "toque humano", ou seja, criar conteúdo com significado profundo, intenção autoral forte ou nuances emocionais complexas que são frequentemente associadas ao design manual (Lazaridis; Fragulis, 2024).

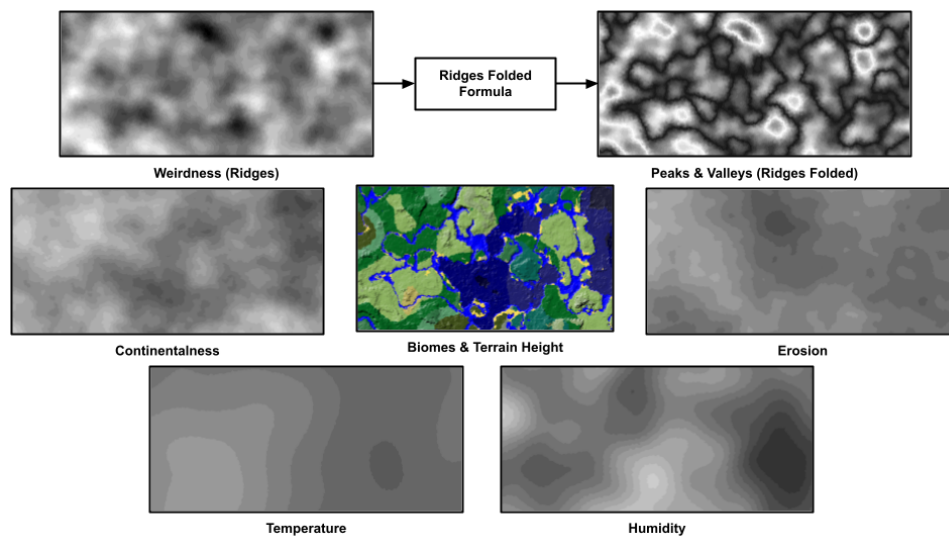
2.2.4 Técnicas Comuns de PCG

O campo da PCG abrange uma diversidade de técnicas algorítmicas, cada uma com suas particularidades e aplicabilidades. Para a geração de níveis, foco deste trabalho, e também para narrativas simples, algumas abordagens se destacam antes da introdução de LLMs, que serão discutidos posteriormente. É importante ressaltar que muitas dessas técnicas podem ser combinadas para alcançar resultados mais complexos e controláveis.

Algoritmos baseados em ruído, como *Perlin Noise* (Perlin, 1985) e *Simplex Noise*

(Perlin, 2001), são frequentemente utilizados para gerar terrenos, texturas, mapas de altura e outras distribuições espaciais que requerem uma aparência orgânica e natural. Esses algoritmos produzem padrões pseudoaleatórios que podem ser parametrizados para controlar características como frequência e amplitude, resultando em formações geográficas verossímeis ou distribuições de recursos. Um exemplo prático disso pode ser observado na Figura 4, que ilustra as funções de densidade (com *3D noise*) utilizadas na geração de biomas e altura de terreno no jogo *Minecraft*.

Figura 4 – Exemplo de funções de densidade que utilizam ruído 3D para geração de biomas e altura de terreno no *Minecraft*

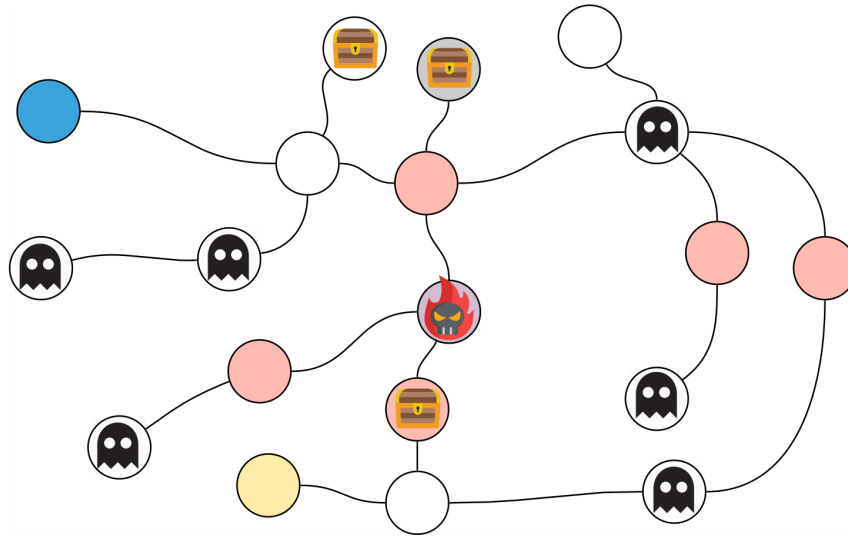


Fonte: Disponível em: https://minecraft.wiki/w/World_generation. Acesso em: 23 jan. 2026.

Gramáticas formais, incluindo *L-Systems* (Lindenmayer, 1968), *Shape Grammars* (Stiny; Gips, 1972) e *Graph Grammars* (Ehrig *et al.*, 1973), oferecem um meio poderoso para definir regras de construção que podem gerar estruturas complexas e recursivas. *L-Systems* são notáveis para modelar o crescimento de plantas e fractais. *Shape Grammars* operam sobre formas geométricas, permitindo a criação de estilos arquitetônicos ou layouts. *Graph Grammars*, particularmente relevantes para a geração de níveis em jogos do tipo *roguelike*, manipulam grafos para definir a estrutura e conectividade de salas, corredores e áreas em um mapa, como pode ser observado na Figura 5.

Técnicas baseadas em agentes e simulação permitem que entidades autônomas "esculpam" o ambiente ou distribuam elementos a partir de suas interações, criando padrões complexos. Um exemplo clássico é o algoritmo *Random Walk* (Pearson, 1905), no qual um agente se move aleatoriamente por uma grade, "cavando" túneis ou caminhos, frequentemente usado na geração de cavernas ou masmorras simples (Koesnaedi; Istiono, 2022). Simulações

Figura 5 – Exemplo de mapa gerado utilizando gramáticas de grafos



Fonte: (Souza *et al.*, 2019)

mais complexas podem envolver múltiplos agentes com comportamentos distintos para criar ecossistemas ou padrões urbanos.

A PCG *Search-Based* (Togelius *et al.*, 2011b) utiliza algoritmos de otimização, como algoritmos genéticos, evolução diferencial ou *simulated annealing*, para explorar o vasto espaço de possíveis conteúdos e encontrar aqueles que melhor satisfazem um conjunto de critérios de qualidade ou restrições de design, definidos por uma função de avaliação. Esta abordagem é útil quando o designer tem objetivos claros sobre as propriedades desejadas do conteúdo, mas a geração direta é complexa.

Finalmente, abordagens baseadas em Aprendizado de Máquina, anteriores à proeminência dos LLMs, já exploravam a capacidade de aprender padrões a partir de dados de exemplo para gerar novo conteúdo. Isso inclui técnicas como *Artificial Neural Networks* (ANN) (McCulloch; Pitts, 1943), Árvore de Decisão (Loh, 2011), ou Cadeias de Markov (Levin; Peres, 2017) para gerar sequências, como layouts de níveis, trechos de música ou comportamento de inimigos. A ascensão dos LLMs pode ser vista como uma evolução natural e mais poderosa desta vertente, especialmente para conteúdo com estrutura sequencial ou semântica complexa, como narrativas.

2.3 LLMs

Os LLMs representam uma classe de modelos de inteligência artificial que demonstraram capacidades notáveis no processamento e geração de linguagem natural. Sua ascensão

tem sido impulsionada por avanços em arquiteturas de redes neurais, disponibilidade de grandes volumes de dados textuais e poder computacional crescente.

2.3.1 Definição e Evolução Histórica dos LLMs

Os LLMs são modelos de inteligência artificial treinados em vastas quantidades de dados textuais, o que lhes confere a capacidade de entender, gerar e manipular a linguagem humana de forma coerente e contextualmente relevante (Hadi *et al.*, 2023). Eles são projetados para prever a próxima palavra ou caractere em uma dada sequência de texto, uma tarefa fundamental na *Natural Language Processing* (NLP).

A evolução dos LLMs pode ser observada desde os primeiros modelos de linguagem estatísticos, como *n-grams* (Brown *et al.*, 1992), que se baseavam em contagens de frequência de sequências de palavras. Posteriormente, as *Recurrent Neural Network* (RNN) e suas variantes, como *Long Short-Term Memory* (LSTM) (Hochreiter; Schmidhuber, 1997), trouxeram melhorias ao capturar dependências de longo alcance no texto, embora ainda enfrentassem desafios com sequências muito longas e paralelização do treinamento (Hadi *et al.*, 2023).

Um marco fundamental nessa evolução foi a introdução da arquitetura Transformer no artigo "*Attention is All You Need*" por Vaswani *et al.* (2017). Essa arquitetura, baseada unicamente em mecanismos de atenção, dispensou a recorrência e permitiu um treinamento mais eficiente e paralelizável em grandes volumes de dados. A partir dos Transformers, surgiram modelos influentes como *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin *et al.*, 2019) e a série GPT (Radford *et al.*, 2018), que estabeleceram novos paradigmas no NLP e impulsionaram o desenvolvimento dos LLMs modernos (Hadi *et al.*, 2023).

2.3.2 Transformers e Mecanismo de Atenção

A arquitetura *Transformer*, conforme introduzida por Vaswani *et al.* (2017), é a base da maioria dos LLMs contemporâneos. Diferentemente das arquiteturas baseadas em RNN ou *Convolutional Neural Networks* (CNN) (LeCun *et al.*, 1998), os Transformers processam todos os *tokens*¹⁰ de uma sequência de entrada simultaneamente, o que facilita a paralelização e o treinamento em hardware especializado como *Graphics Processing Unit* (GPU) e *Tensor Processing Unit* (TPU) (Lin *et al.*, 2021).

¹⁰ Um *token* pode ser uma palavra, uma subpalavra (um subconjunto de uma palavra) ou até mesmo um único caractere. Fonte (Acesso em: 23 jan. 2026): <https://developers.google.com/machine-learning/crash-course/llm/transformers>

Um dos seus componentes mais revolucionários é o mecanismo de *self-attention*. Este mecanismo permite que o modelo, ao processar uma *token* em uma sequência, pondere a importância de todos os outros *tokens* na mesma sequência. Essencialmente, ele calcula "scores de atenção" que determinam o quanto cada palavra deve contribuir para a representação da palavra atual. Isso possibilita que o modelo capture dependências de longo alcance e entenda o contexto de forma mais eficaz, mesmo em sequências de texto extensas (Vaswani *et al.*, 2017; Naveed *et al.*, 2024). A capacidade de processar sequências em paralelo, combinada com o poder do mecanismo de auto-atenção para modelar relações contextuais complexas, tornou os *Transformers* particularmente adequados para o treinamento em grandes volumes de dados textuais, resultando nas capacidades impressionantes dos LLMs atuais (Lin *et al.*, 2021).

2.3.3 *Treinamento e Capacidades dos LLMs*

O treinamento de LLMs geralmente envolve duas fases principais. A primeira é o pré-treinamento, no qual o modelo é exposto a enormes quantidades de dados textuais não rotulados (tais como, livros, artigos, websites). Durante essa fase, o modelo aprende a prever palavras mascaradas em uma sentença (como no BERT) ou a próxima palavra em uma sequência (como no GPT), o que o força a aprender padrões linguísticos, conhecimento de mundo e relações semânticas complexas (Liu *et al.*, 2024; Naveed *et al.*, 2024).

Após o pré-treinamento, os modelos podem passar por um processo de ajuste *fine-tuning* para tarefas específicas. Nesta etapa, o modelo pré-treinado é treinado adicionalmente em um dataset menor e específico da tarefa (por exemplo, tradução, sumarização, resposta a perguntas), com dados rotulados. Isso adapta o conhecimento geral adquirido durante o pré-treinamento para a tarefa desejada (Liu *et al.*, 2024).

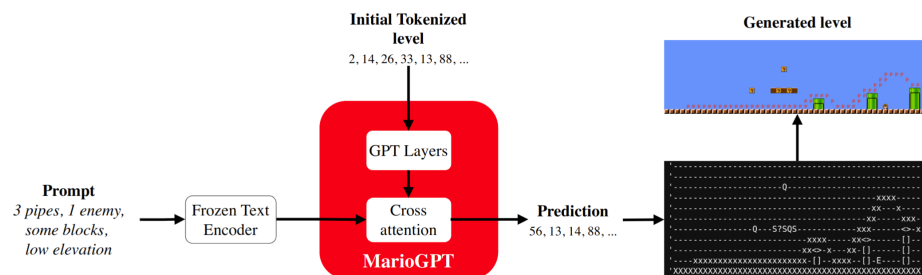
Como resultado, os LLMs exibem uma vasta gama de capacidades, incluindo a geração de texto coerente e criativo, tradução entre idiomas, sumarização de documentos longos, resposta a perguntas de forma informativa, e até mesmo a escrita de código (Hadi *et al.*, 2023). Uma característica notável é a emergência de capacidades complexas, como o raciocínio em múltiplos passos e o aprendizado em poucos exemplos (*few-shot learning*) ou mesmo sem exemplos prévios (*zero-shot learning*), no qual o modelo pode realizar tarefas para as quais não foi explicitamente treinado, apenas com base em uma descrição da tarefa e, opcionalmente, alguns exemplos fornecidos no prompt (Vaswani *et al.*, 2017; Naveed *et al.*, 2024).

2.3.4 LLMs Aplicados à PCG

As capacidades dos LLMs, especialmente sua proficiência na compreensão e geração de linguagem natural, alinham-se de forma promissora com as necessidades da PCG em jogos (Maleki; Zhao, 2024). A PCG visa criar conteúdo de jogo (níveis, narrativas, itens, etc.) algoritmicamente, e os LLMs oferecem novas possibilidades para enriquecer e personalizar esse processo.

Um dos potenciais mais diretos dos LLMs na PCG é a geração de elementos textuais. Isso inclui a criação de narrativas dinâmicas que se adaptam às ações do jogador, diálogos interativos e realistas para NPCs, descrições ricas e detalhadas de itens, locais e personagens, e histórias de fundo para o mundo do jogo (Rolim, 2023). A capacidade dos LLMs de manter a coerência e o contexto ao longo de textos extensos é particularmente valiosa para esses fins.

Figura 6 – Arquitetura do sistema *MarioGPT*. Um prompt textual é processado por um encoder de texto, cujas saídas são usadas em um mecanismo de atenção cruzada com as camadas GPT. As camadas GPT também recebem um nível "tokenizado" inicial. O sistema então prediz a continuação do nível "tokenizado", que é convertido para um nível de jogo visual.



Fonte: (Sudhakaran *et al.*, 2023)

Além de elementos puramente textuais, os LLMs também têm potencial para influenciar a geração de elementos estruturais, ainda que indiretamente. Por exemplo, um LLM pode gerar descrições textuais detalhadas de *layouts* de níveis, sequências de eventos de uma missão, ou regras de um sistema de jogo. Essas descrições podem então ser interpretadas por outros sistemas ou algoritmos de PCG para traduzi-las em conteúdo de jogo concreto (Maleki; Zhao, 2024). Pesquisas recentes, como o projeto *MarioGPT* (cujas arquitetura é ilustrada na Figura 6), exploraram o uso de LLMs para gerar níveis de jogos 2D a partir de *prompts* textuais, indicando um caminho para a personalização de conteúdo de jogo com base nas preferências do usuário expressas em linguagem natural.

3 TRABALHOS RELACIONADOS

A utilização de Modelos de LLM na PCG em jogos é uma área de pesquisa em expansão. Diversos trabalhos têm explorado o potencial dos LLMs para criar não apenas elementos de jogo isolados, mas também mundos e narrativas coesas. Nesta seção, são apresentados trabalhos relevantes que abordam a geração de conteúdo de jogos com LLMs, estabelecendo um paralelo com a proposta deste TCC e contextualizando suas contribuições. A análise segue uma ordem decrescente de afinidade, partindo do trabalho mais diretamente relacionado até os que apresentam semelhanças em aspectos mais específicos da abordagem.

O artigo de Nasir *et al.* (2024), intitulado *Word2World*, apresenta uma abordagem que se assemelha de forma significativa aos objetivos deste TCC. Os autores propõem um sistema que utiliza LLMs para gerar mundos de jogo 2D jogáveis a partir de uma história inicial. O processo é realizado em um *pipeline* multifásico que inclui a criação da história, a extração de informações cruciais (como personagens, *tiles*¹ e objetivos), a geração do mundo e um ciclo de *feedback* para avaliação e refinamento. A principal semelhança com o presente trabalho reside no objetivo central de conectar uma narrativa de alto nível à geração de um nível de jogo concreto e coerente, utilizando um LLM para mediar ambas as tarefas. A principal diferença, contudo, está no escopo de aplicação. Enquanto *Word2World* foca em jogos 2D de visão superior de forma genérica, este TCC se especializa no gênero *roguelike*, que possui restrições e características próprias, como a alta rejogabilidade e a estrutura de masmorras, aspectos que exigem uma adaptação e especialização do processo de geração.

De forma similar, o artigo *MarioGPT*, de Sudhakaran *et al.* (2023), demonstra a viabilidade da geração de níveis a partir de descrições textuais, focando no gênero de plataforma. O sistema utiliza um modelo GPT-2 ajustado para criar fases do jogo *Super Mario Bros*² a partir de *prompts* simples, como "muitos canos, poucos inimigos, elevação baixa". A semelhança com este TCC reside na abordagem "*text-to-level*", provando que LLMs podem interpretar descrições e produzir conteúdo estruturado e funcional para um jogo. No entanto, a distinção fundamental está no gênero e na complexidade estrutural. Enquanto os níveis de plataforma são tipicamente lineares, os mapas de *roguelike* exigem uma estrutura de grafo com salas interconectadas, um desafio de conectividade e *layout* que não é o foco principal de *MarioGPT*.

¹ *tiles* (ou ladrilhos, em português) são pequenas imagens ou gráficos que são usados para construir o cenário ou mapa do jogo

² Site oficial do *Super Mario Bros* (Acesso em: 23 jan. 2026): <https://www.nintendo.com/pt-pt/Jogos/NES/Super-Mario-Bros-803853.html>

A dissertação de mestrado de Auxtero (2023) propõe o sistema *Game Environment Design Creator* (GEDC), explorando a criação de um sistema para o design de ambientes de jogos, utilizando geração procedural com inteligência artificial para facilitar a criação de narrativas interativas. O trabalho foca em desenvolver um sistema intuitivo que permite aos usuários criar e experienciar diferentes histórias, automatizando a geração inicial do ambiente, mas permitindo customização posterior. A semelhança com este TCC está na aplicação de LLMs para a geração automática de ambientes de jogo e no objetivo de criar uma ferramenta acessível para criadores. A diferença reside no foco: o trabalho de Auxtero (2023) é mais abrangente, visando jogos de aventura em geral e com grande ênfase na interface de customização manual, enquanto este TCC se concentra nas restrições e particularidades algorítmicas específicas do gênero *roguelike*.

Por fim, em um artigo publicado na *2024 IEEE Conference on Games* (CoG), Gallotta *et al.* (2024a) exploram a geração consistente de conteúdo através de um *framework* chamado *LLMaker*. A principal contribuição do trabalho é o uso de *function calling*, no qual o LLM atua como um tradutor de comandos em linguagem natural para chamadas de funções formais em um sistema de *back-end*. Essa abordagem garante que toda modificação no conteúdo do jogo (neste caso, *layouts* de masmorras) seja válida e adira às restrições do domínio, mitigando o problema de alucinações e inconsistências do LLM. A semelhança com este TCC está no reconhecimento do desafio da consistência na geração de conteúdo e na busca por uma solução que garanta a validade dos artefatos gerados. O domínio de aplicação, focado em *dungeon crawlers*, também é bastante próximo ao gênero *roguelike*. A diferença principal, entretanto, está no paradigma de interação. *LLMaker* é uma ferramenta de *codesign* iterativo, na qual um designer humano interage continuamente com o sistema via *chat*, enquanto a proposta deste TCC foca em um processo mais automatizado de geração inicial a partir de uma entrada textual.

O Quadro 1 apresenta uma análise comparativa dos trabalhos relacionados mais próximos. A comparação é baseada em critérios relevantes para esta pesquisa, destacando as semelhanças e diferenças entre as abordagens.

Quadro 1 – Comparativo dos Trabalhos Relacionados

Trabalho	Geração de níveis a partir de descrições textuais	Foco em engenharia de prompt	Saída em formato estruturado (e.g., JSON)	Abordagem focada no gênero Roguelike
Word2World (Nasir <i>et al.</i> , 2024)	x	x		
MarioGPT (Sudhakaran <i>et al.</i> , 2023)	x			
GEDC (Auxtero, 2023)	x		x	
LLMaker (Gallotta <i>et al.</i> , 2024a)	x	x		
Trabalho Proposto	x	x	x	x

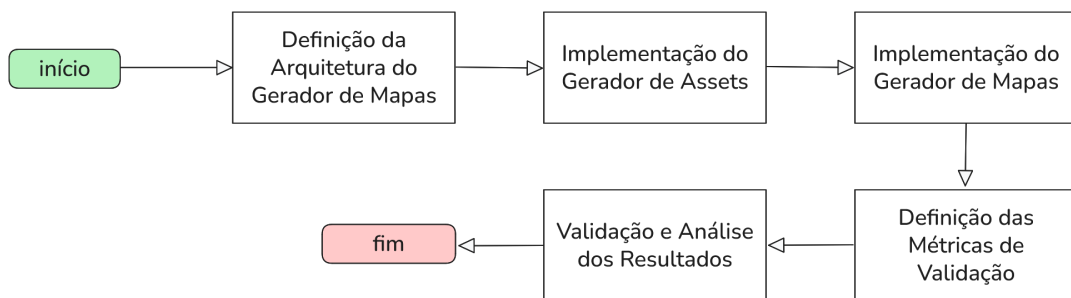
Fonte: Elaborado pelo autor

4 METODOLOGIA

4.1 Visão Geral da Metodologia

A Figura 7 ilustra a sequência de fases que orientam a pesquisa, desde a concepção teórica até a análise empírica dos resultados. Este percurso metodológico assegura a rastreabilidade e a reprodutibilidade dos procedimentos adotados para desenvolver e validar o sistema de geração de mapas para jogos *roguelike*.

Figura 7 – Fluxograma das etapas metodológicas do trabalho.



Fonte: Elaborado pelo autor

4.2 Definição da Arquitetura do Gerador de Mapas

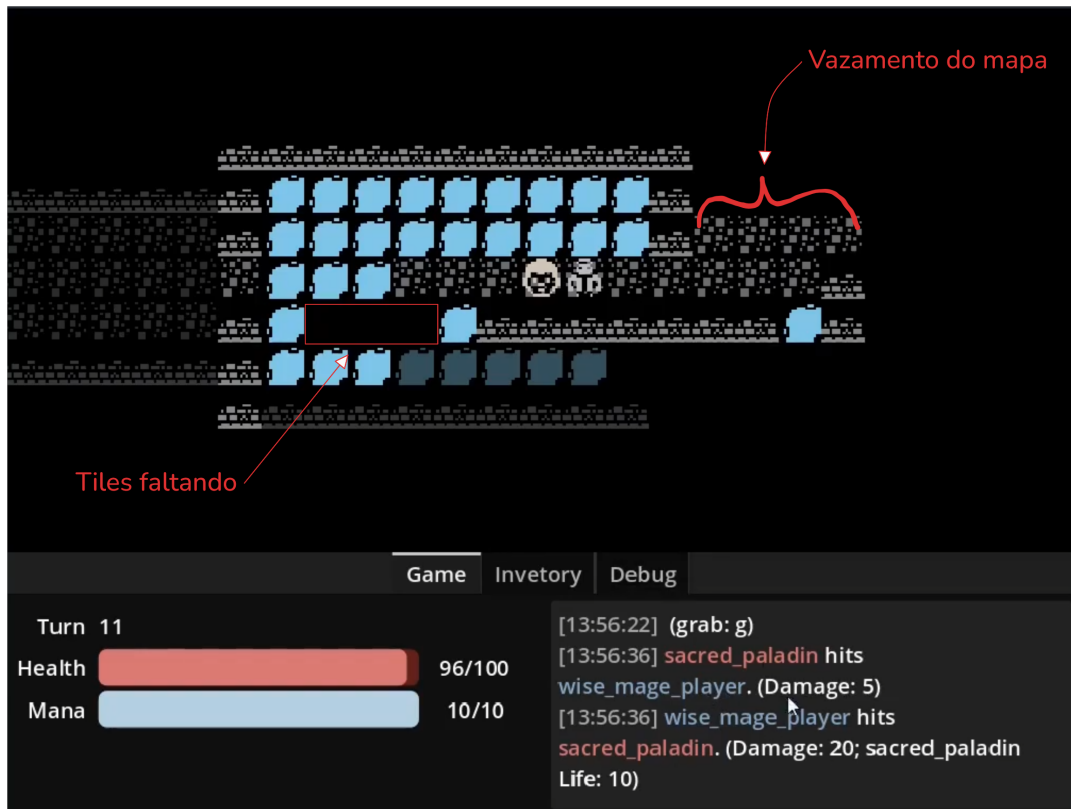
A concepção da arquitetura deste sistema parte da identificação das limitações intrínsecas aos LLMs no que tange ao processamento de informações geométricas e espaciais. Embora as LLMs demonstrem capacidades excepcionais na geração de texto e raciocínio semântico, o raciocínio espacial permanece um desafio significativo.

Conforme demonstrado por Yan *et al.*, modelos de estado da arte, como o *GPT-4*, apresentam dificuldades substanciais em tarefas que exigem compreensão precisa de coordenadas, como plotar pontos em espaços 2D/3D ou executar algoritmos de busca de caminho. Esta limitação é corroborada por Gallotta *et al.*, que argumentam que, apesar do potencial das LLMs como assistentes de design, a falta de raciocínio espacial impede que estes modelos gerem, de forma autônoma e consistente, *layouts* de níveis complexos que sejam jogáveis e estruturalmente coerentes, requisitos fundamentais para o gênero *roguelike*.

Durante a fase exploratória desta pesquisa, experimentos práticos corroboraram essas dificuldades teóricas. Tentativas de utilizar o LLM para gerar diretamente a topologia e a disposição dos elementos resultaram em mapas geometricamente inválidos. A Figura 8 ilustra

um caso específico onde o modelo falhou em manter a integridade da sala, resultando em *tiles* faltando no interior da estrutura e no "vazamento" de partes do mapa para áreas desconexas. Tais erros evidenciam que a LLM, quando encarregada da estrutura espacial, tende a alucinar posições, comprometendo a jogabilidade.

Figura 8 – Exemplo de falha na geração direta de topologia por LLM. Observam-se inconsistências estruturais graves, destacadas como *tiles* ausentes no chão da sala e vazamento de estruturas para fora dos limites lógicos do mapa.



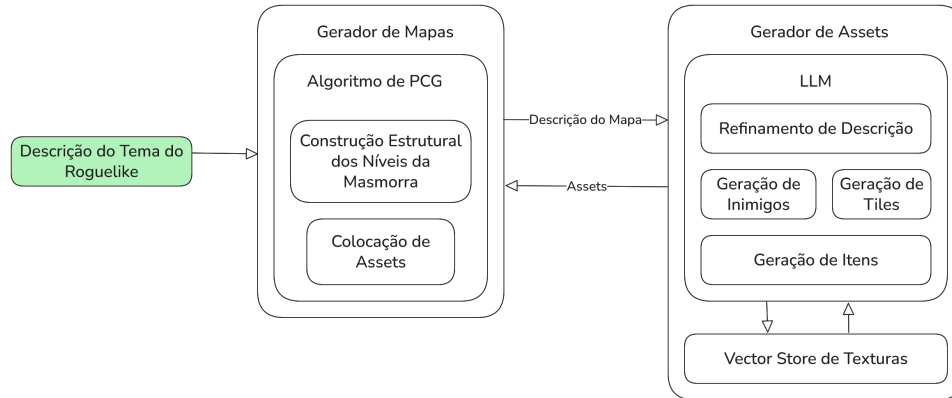
Fonte: Elaborado pelo autor

Diante deste cenário e das falhas observadas, este trabalho propõe uma arquitetura híbrida que desacopla a responsabilidade criativa da responsabilidade estrutural. A abordagem divide o processo de geração em dois módulos distintos e complementares: o Gerador de *Assets*, impulsionado por LLM, e o Gerador de Mapas, fundamentado em algoritmos de PCG tradicionais. A visão geral desta metodologia e o fluxo de dados entre os componentes são apresentados na Figura 9.

4.2.1 Geração de *Assets* via LLM

Nesta etapa, a LLM é utilizada para tarefas nas quais sua proficiência semântica e criativa é maximizada. O modelo recebe como entrada o tema do *roguelike* e é responsável por

Figura 9 – Arquitetura Híbrida proposta: O fluxo inicia com a descrição do tema, que alimenta tanto o algoritmo de PCG quanto a LLM. A LLM gera assets semânticos (inimigos, itens e tiles) utilizando um Vector Store de texturas (RAG), enquanto o algoritmo de PCG utiliza esses assets para construir a estrutura espacial da masmorra.



Fonte: Elaborado pelo autor

gerar descrições detalhadas e instanciar os *assets* do jogo. Neste contexto, definem-se *assets* como as entidades que povoam o mapa: inimigos e itens consumíveis ou equipáveis. A LLM, portanto, enriquece o ambiente com contexto narrativo e visual, sem a necessidade de gerenciar a topologia global da masmorra.

4.2.2 Construção Estrutural via PCG Tradicional

A segunda etapa assume a responsabilidade pela disposição espacial dos elementos gerados anteriormente. Algoritmos clássicos de PCG, baseados em geração de números pseudo-aleatórios, são empregados para definir a geometria das salas, a conectividade dos corredores e o posicionamento lógico dos *assets*. O algoritmo recebe a lista de objetos criados pela LLM e os distribui pelo mapa garantindo a validade topológica e a navegabilidade, contornando as limitações espaciais das LLMs apontadas por Yan *et al.*.

Esta abordagem híbrida oferece duas vantagens principais. Primeiramente, a **eficiência computacional**: delegar a estrutura do mapa para algoritmos determinísticos ou estocásticos clássicos é significativamente menos custoso do que realizar inferências complexas em uma LLM para cada célula de uma grade. Em segundo lugar, a arquitetura maximiza a **rejugabilidade**. Uma única execução do Gerador de *Assets* pode produzir um conjunto rico de entidades temáticas que, por sua vez, podem ser recombinadas pelo algoritmo de PCG para gerar uma quantidade virtualmente infinita de variações de mapas distintos, preservando a essência de exploração e imprevisibilidade característica dos jogos *roguelike*.

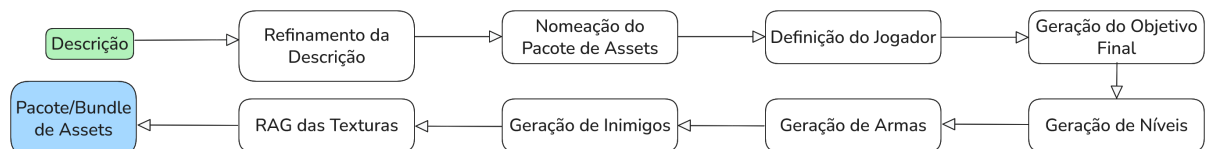
4.3 Implementação do Gerador de Assets

A materialização de um jogo *roguelike* requer não apenas a definição de regras estruturais, mas também a criação de elementos tangíveis que povoam o mundo virtual. Para garantir que inimigos, itens e ambientes compartilhem uma identidade visual e narrativa coesa, o sistema não gera ativos isoladamente, mas sim um objeto completo denominado *Asset Bundle*.

O *pipeline* de geração opera através de uma cadeia de requisições sequenciais, desenhada para isolar o contexto e maximizar a precisão do LLM em cada tarefa específica. Conforme ilustrado na Figura 10, o fluxo de execução para a criação de um *bundle* completo consiste em oito etapas principais:

1. **Descrição:** Entrada inicial do usuário (tema ou ideia).
2. **Refinamento da Descrição:** Enriquecimento da descrição inicial do usuário para garantir maior detalhamento semântico.
3. **Nomeação do Pacote de Assets:** Geração de um título temático para o pacote de ativos.
4. **Definição do Jogador:** Criação da classe, história de fundo e aparência do avatar.
5. **Geração do Objetivo Final:** Definição do artefato de vitória e sua justificativa narrativa.
6. **Geração de Níveis:** Criação estruturada dos biomas e conjuntos de *tiles* para cada profundidade da masmorra.
7. **Geração de Armas:** Instanciação dos itens ofensivos com atributos de raridade e peso.
8. **Geração de Inimigos:** Criação das entidades hostis com atributos de ameaça e peso.
9. **RAG das Texturas:** Recuperação da textura em *pixel art* mais semanticamente compatível no banco vetorial para cada entidade gerada nos passos anteriores.

Figura 10 – Fluxo de geração do Pacote de Assets: da descrição inicial à instanciação dos objetos via RAG.



Fonte: Elaborado pelo autor

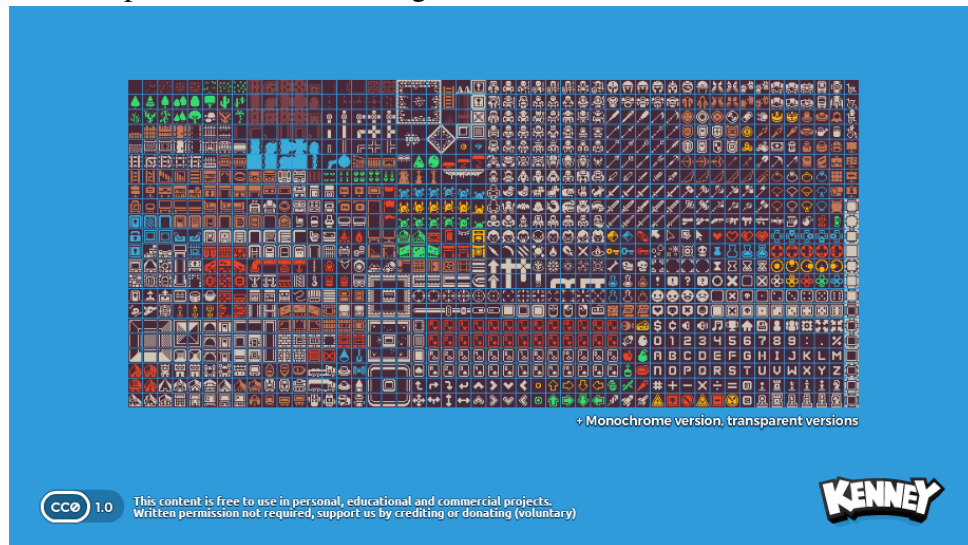
O resultado final deste processo é um objeto JSON estruturado que contém todas as definições necessárias para que o Gerador de Mapas (descrito na Seção 4.4) construa os níveis, popule as salas com inimigos balanceados e distribua os itens de acordo com a progressão de dificuldade.

Para associar a capacidade gerativa dos LLMs com representações visuais concretas adotou-se a técnica de RAG. Esta abordagem permite recuperar texturas específicas de um banco vetorial com base em similaridade semântica, uma estratégia validada recentemente na literatura por Nasir *et al.*, que utilizam LLMs para gerar mundos 2D completos integrando geração procedural e recuperação de ativos.

4.3.1 Dataset e Pré-processamento Visual

A base visual do sistema foi construída a partir do "*1-Bit Pack*"¹ disponibilizado pelo site *Kenney* sob licença *Creative Commons CC0*. A escolha deste conjunto deve-se à sua alta diversidade temática e consistência estilística, ideal para a prototipagem rápida de jogos *roguelike*. O pacote original contém 1078 *tiles* de 16×16 pixels, abrangendo desde elementos de interface de usuário até vegetação e arquitetura, conforme ilustrado na Figura 11.

Figura 11 – Tilemap completo do pacote "*1-Bit Pack*" do site *Kenney*, contendo a totalidade dos ativos gráficos disponíveis antes da filtragem.



Fonte: Disponível em: <https://kenney.nl/assets/1-bit-pack>. Acesso em: 23 jan. 2026.

Para garantir a precisão da busca vetorial e evitar a recuperação de elementos inadequados, foi realizado um processo de curadoria e anotação manual. O *dataset* foi filtrado e segmentado em três categorias macro, totalizando 489 descrições textuais associadas a seus respectivos arquivos de imagem em base64:

- **Entities** (88 descrições): Inclui inimigos, NPCs e o avatar do jogador.
- **Items** (183 descrições): Abrange poções, armas, armaduras, chaves e tesouros.

¹ *Kenney 1-Bit Pack* (Acesso em: 23 jan. 2026): <https://kenney.nl/assets/1-bit-pack>

- **Environments** (218 descrições): Contém paredes, pisos, obstáculos naturais e vegetação.

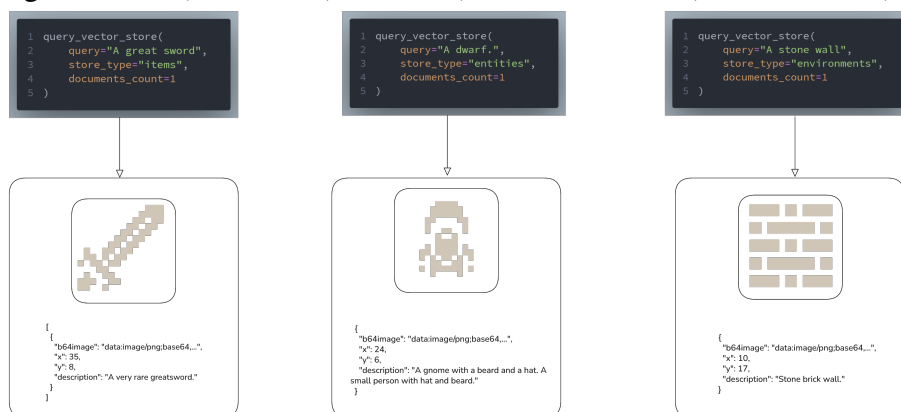
Esta segmentação é crucial para evitar alucinações de categoria durante a recuperação. Por exemplo, ao solicitar "uma espada longa", o sistema deve restringir a busca ao subconjunto de *Items*, eliminando a possibilidade de retornar uma textura de parede ou chão que pudesse ter uma descrição vetorial semanticamente próxima em um espaço latente não supervisionado.

4.3.2 Stack Tecnológico e Busca Vetorial

A implementação foi realizada na linguagem *Python*, escolhida devido ao seu vasto ecossistema de bibliotecas voltadas para Inteligência Artificial. A orquestração do fluxo de dados utiliza o *framework LangChain*, que facilita a construção de agentes adaptáveis e a integração com modelos de linguagem. Para a persistência e recuperação dos *embeddings* das descrições dos *assets*, utilizou-se o *ChromaDB*, um banco de dados vetorial *open-source* otimizado para busca semântica.

O núcleo do mecanismo de recuperação reside na função `query_vector_store`. Esta função aceita três parâmetros principais: a consulta textual (`query`), o tipo de armazenamento (`store_type`) para filtragem de categoria, e a quantidade de documentos a recuperar (`documents_count`), fixada em $n = 1$ para esta aplicação. O ChromaDB foi configurado para utilizar a Distância Euclidiana ao Quadrado (*Squared L2*) como métrica de similaridade, retornando o ativo cuja descrição vetorial apresenta a menor distância em relação ao vetor da consulta gerada pela LLM. A Figura 12 demonstra o funcionamento prático deste módulo.

Figura 12 – Exemplos de execução da função de busca vetorial para três categorias distintas: Items ("A great sword"), Entities ("A dwarf") e Environments ("A stone wall").



Fonte: Elaborado pelo autor

4.3.3 Geração de Entidades com Saídas Estruturadas

Um dos desafios recorrentes na utilização de LLMs em sistemas de software é a natureza não determinística e não estruturada das saídas textuais. Para mitigar esse problema e garantir que o Gerador de Assets produza objetos de jogo válidos, empregou-se o conceito de *Structured Outputs*. Conforme discutido por Shorten *et al.*, a capacidade de forçar modelos de linguagem a seguir esquemas rígidos, como JSON, é essencial para o desenvolvimento de sistemas de IA compostos.

Neste trabalho, a integração entre *LangChain* e a biblioteca *Pydantic* foi utilizada para definir esquemas de dados rigorosos. Antes de invocar a LLM, define-se uma classe *Pydantic* que representa o objeto de jogo desejado (por exemplo, *MeleeWeapon*). A LLM é então instruída a preencher os campos desta classe com base na descrição temática fornecida pelo usuário. O resultado é um objeto estruturado contendo não apenas os metadados do item, mas também a textura recuperada via RAG.

4.3.4 Abstração de Design e Valores Discretos

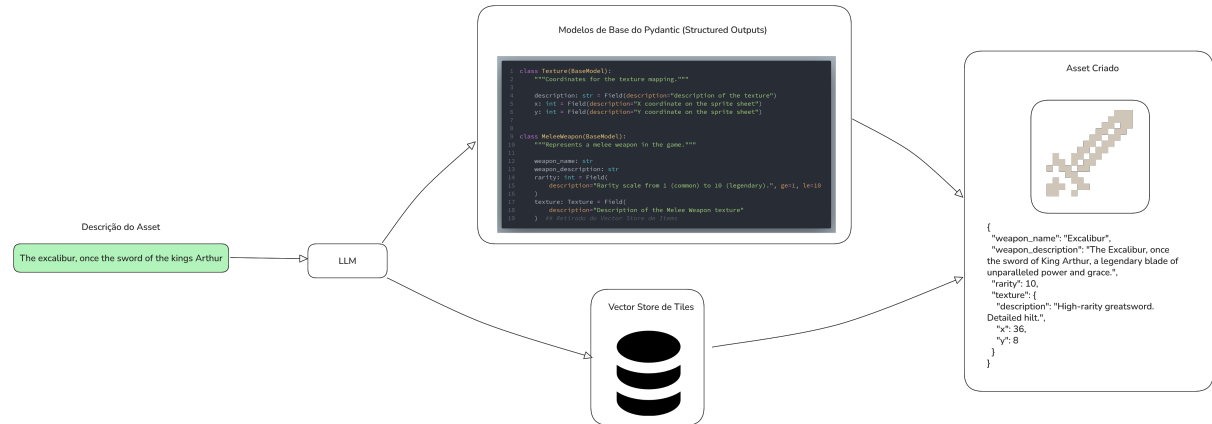
Uma decisão de design fundamental na implementação foi a abstração dos atributos numéricos dos objetos de jogo. Em vez de solicitar à LLM que determine valores brutos de dano ou pontos de vida (o que exigiria que o modelo mantivesse um contexto complexo sobre o balanceamento matemático do jogo), optou-se pelo uso de valores discretos em uma escala de 0 a 10 para atributos como *rarity* (raridade), *threat* (ameaça) e *weight* (peso).

Esta abordagem, ilustrada na Figura 13, desacopla a geração criativa do balanceamento mecânico. A LLM foca na coerência narrativa e semântica (por exemplo, uma "Excalibur" deve ter raridade alta), enquanto o motor do jogo traduz esses valores discretos em estatísticas finais (dano, vida, armadura) através de multiplicadores definidos em arquivos de configuração.

4.3.5 Interface de Comunicação

Para possibilitar a integração modular com o Gerador de Mapas e com a interface do usuário, o Gerador de Assets foi encapsulado e exposto através de uma API *Representational State Transfer* (REST) desenvolvida com *FastAPI*. A escolha desta arquitetura justifica-se pela nativa compatibilidade com o formato JSON gerado pelos modelos *Pydantic*, permitindo uma comunicação fluida e assíncrona entre os componentes do sistema.

Figura 13 – Fluxo de geração de um asset estruturado: a descrição textual é processada, combinada com a textura recuperada do Vector Store e validada via Pydantic para criar um objeto de jogo funcional com atributos discretos.



Fonte: Elaborado pelo autor

4.4 Implementação do Gerador de Mapas

A segunda etapa da arquitetura híbrida proposta consiste no desenvolvimento do Gerador de Mapas. Enquanto o módulo anterior é responsável pela criatividade visual, este módulo assume a responsabilidade pela lógica espacial, validação topológica e pela orquestração do ciclo de jogo.

4.4.1 Escolha Tecnológica e Arquitetura Base

Para a implementação do sistema, adotou-se a *game engine Godot* em sua versão 4.5.1. A escolha justifica-se pela natureza *open-source* da ferramenta, que favorece a reprodutibilidade acadêmica e a transparência na implementação de algoritmos. Segundo Salmela (2022), o *Godot* oferece uma integração robusta entre sua linguagem de *scripting* nativa (*GDScript*) e o sistema de nós, facilitando o desenvolvimento de arquiteturas modulares.

A implementação baseou-se na adaptação do tutorial "*Complete Roguelike Tutorial*"², originalmente desenvolvido para a biblioteca *libtcod* em *Python*, hospedado no wiki comunitária *RogueBasin*. A lógica original de gerenciamento de turnos e Campo de Visão (*Field of View* (FOV)) foi portada para a arquitetura de nós do *Godot*, aproveitando recursos nativos como:

- **RandomNumberGenerator:** Para garantir a estocasticidade determinística através do controle de *seeds*, essencial para a PCG.
- **HTTPRequest:** Utilizado para comunicar-se de forma assíncrona com a API do Gerador

² "*Complete Roguelike Tutorial*" (Acesso em: 23 jan. 2026): https://www.roguebasin.com/index.php/Complete_Roguelike_Tutorial,_using_python%2Blibtcod

de Assets desenvolvido na Seção 4.3, recebendo os objetos em formato JSON.

4.4.2 Integração de Dados e Desserialização

O desafio central desta etapa foi converter as descrições textuais e parâmetros discretos gerados pela LLM em objetos de jogo funcionais. Para isso, foi implementada uma classe base denominada `Serializable`. Todos os objetos dinâmicos do jogo (inimigos e itens) herdam desta classe.

A classe expõe o método `load(data: Dictionary)`, que atua como um intérprete. Ao receber o JSON da etapa anterior, este método instancia o nó correspondente, aplica a textura recuperada via RAG e atribui os valores de atributos (como vida, dano e defesa) baseados nos valores discretos de *threat*, *rarity* e *weight* definidos pela LLM.

4.4.3 Algoritmo de Geração Estrutural

Para a construção da topologia da masmorra, optou-se por um algoritmo de Alocação de Retângulos Aleatórios com verificação de sobreposição. Esta técnica garante salas estruturadas e conectadas, fundamentais para a navegação tática em jogos *roguelike*.

O algoritmo opera iterativamente tentando alocar salas dentro de um espaço delimitado. Se uma nova sala não interceptar nenhuma das existentes, ela é "esculpida" no mapa. A conectividade é garantida ligando-se o centro da nova sala ao centro da sala anteriormente criada através de corredores em formato de "L". O Algoritmo 1 apresenta o pseudocódigo da lógica implementada no método `generate_basic_dungeon`.

Algoritmo 1: Geração Procedural da Estrutura da Masmorra

Entrada: Restrições Geométricas (D_{mapa}), Limite de Tentativas (N_{max})

Saída: Topologia da Masmorra ($Salas$)

Início

```

 $Salas \leftarrow \emptyset;$ 
para  $i \leftarrow 0$  até  $N_{max}$  faça
     $Candidata \leftarrow \text{GerarRetanguloAleatorio}(D_{mapa});$ 
    se  $\text{HaIntersecao}(Candidata, Salas)$  então
        continuar;
    fim
     $\text{EscavarNoMapa}(Candidata);$ 
    se  $Salas$  não está vazia então
         $Alvo \leftarrow \text{ObterUltimaSala}(Salas);$ 
         $\text{CriarCorredorConectando}(Alvo, Candidata);$ 
    fim
    Adicionar  $Candidata$  ao conjunto  $Salas;$ 
fim
retorna  $Salas;$ 

```

fim

4.4.4 Povoamento e Progressão de Dificuldade

Após a definição estrutural, o algoritmo popula as salas geradas. A distribuição de entidades segue uma curva de progressão baseada na profundidade da masmorra.

O sistema utiliza os metadados gerados pela LLM na Seção 4.3. Itens e inimigos possuem atributos de *rarity*, *threat*, *weight*. O algoritmo de povoamento calcula um peso probabilístico onde, quanto maior a profundidade do nível, maior a chance de instanciar objetos com valores altos nesses atributos. Escadas são posicionadas na última sala gerada, garantindo que o jogador precise atravessar a estrutura para avançar.

4.4.5 Ciclo de Gameplay e Condição de Vitória

Para validar a integridade do gerador e a coerência dos assets criados, foi implementado um ciclo de jogo completo. O design deste ciclo é inspirado diretamente na estrutura

de objetivos do jogo seminal *Rogue*, conforme detalhado em seu manual oficial (EPYX, INC., 1985), onde o jogador deve recuperar o Amuleto de *Yendor* e retornar à superfície. Nesta implementação, o objetivo definido é descer através de níveis gerados proceduralmente até encontrar o Artefato Final. Este artefato é um item especial, também gerado pela LLM, porém forçado a ter parâmetros de raridade máxima e uma descrição narrativa de importância crítica.

A condição de vitória exige que o jogador colete o artefato e retorne vivo ao primeiro nível da masmorra, invertendo o fluxo de progressão. Em conformidade com as definições teóricas do gênero *roguelike* apresentadas na Seção 2.1 e reforçadas pela estrutura clássica de *Rogue* (EPYX, INC., 1985), o sistema implementa a mecânica de Morte Permanente. Caso os pontos de vida do jogador cheguem a zero em qualquer momento da jornada, todo o progresso (incluindo o mapa explorado, itens coletados e o próprio artefato) é perdido, e o jogo reinicia gerando uma nova *seed* aleatória, resultando em uma nova masmorra, mas utilizando os mesmos assets gerados pela descrição textual.

4.5 Definição das Métricas de Validação

A avaliação de conteúdo gerado proceduralmente, especialmente quando envolve aspectos subjetivos como narrativa e estética, apresenta desafios inerentes que dificultam o uso exclusivo de métricas quantitativas tradicionais. Para mitigar essa questão e alinhar-se ao estado da arte na pesquisa de geração de mundos virtuais, esta metodologia adota a abordagem de *LLM-based Evaluation*, onde modelos de linguagem de larga escala atuam como juízes da qualidade do conteúdo gerado. Esta estratégia é fundamentada nos trabalhos recentes de Nasir *et al.*, que utilizam LLMs para pontuar a coerência entre histórias e mundos 2D, e de Huang, que expandem o conceito para ambientes 3D, aplicando análises de similaridade e reconstrução narrativa. Neste trabalho, foram definidas duas métricas principais para validar a eficácia do *pipeline*: a **Coerência** e a **Reconstrução Semântica**.

4.5.1 Métrica de Coerência (Coherence)

Inspirada diretamente na metodologia proposta pelo sistema *Word2World* de Nasir *et al.*, esta métrica visa quantificar o alinhamento lógico e semântico entre a intenção textual e o artefato de jogo produzido. O processo consiste em fornecer a uma LLM avaliadora dois componentes distintos:

1. A descrição textual aprimorada e enriquecida (gerada na etapa anterior pela LLM);
2. O conjunto de *Assets* gerados em formato JSON (resultado da aplicação de RAG e *Structured Outputs*).

O modelo é instruído a atuar como um crítico e atribuir uma pontuação de 0 a 100, avaliando critérios como: a similaridade entre o ambiente descrito e os elementos visuais selecionados; e a consistência dos inimigos e personagens.

É crucial destacar uma adaptação específica para o contexto deste trabalho: a "descrição textual" fornecida para a avaliação não é a consulta inicial bruta do usuário, mas sim a **descrição aprimorada** gerada pelo Gerador de *Assets*. Como o sistema converte a intenção do usuário em um contexto de jogo *Roguelike* (adicionando detalhes sobre masmorras, itens finais e perigos específicos), a coerência deve ser medida em relação a este texto refinado, garantindo que os objetos concretos do jogo reflitam as convenções do gênero estabelecidas pelo próprio sistema.

4.5.2 Métrica de Reconstrução Semântica

Para complementar a avaliação direta, adota-se uma métrica de consistência cíclica, similar à abordagem de *reconstructed story-based similarity analysis* explorada por Huang. Esta métrica testa a capacidade dos ativos gerados de "comunicarem" seu tema de volta para um observador, sem conhecimento prévio do *prompt* original. O processo de validação ocorre nas seguintes etapas:

1. O conjunto de *tiles* e ativos utilizados para compor o mapa (representados em JSON) é submetido a uma LLM.
2. Solicita-se que o modelo descreva qual seria o tema, a ambientação e o possível enredo do jogo, baseando-se única e exclusivamente nos ativos fornecidos.
3. A nova descrição gerada (texto reconstruído) é comparada com a **descrição aprimorada** original (utilizada na geração dos *Assets*).

A comparação quantitativa é realizada através da geração de *embeddings* para ambos os textos, utilizando a **Similaridade de Cosseno** para mensurar a proximidade semântica no espaço vetorial. Uma similaridade maior (mais próxima de 1) indica que o conjunto de ativos gerado preservou com sucesso as características fundamentais do tema solicitado, a ponto de permitir que uma inteligência artificial deduz a descrição original a partir apenas dos elementos do jogo.

5 RESULTADOS

Este capítulo apresenta os resultados obtidos com a implementação do sistema proposto. A avaliação foi dividida em duas etapas principais: uma análise quantitativa, baseada em métricas automatizadas suportadas por LLMs (*LLM-as-a-Judge*), e uma análise qualitativa, que examina a coerência semântica e visual dos ativos gerados em cenários de teste específicos.

5.1 Configuração Experimental

Para validar a viabilidade e a eficácia do gerador de mapas, foram definidos parâmetros fixos para a geração dos dos assets. Essa padronização é essencial para garantir uma comparação justa entre os diferentes modelos de linguagem avaliados.

O gerador foi configurado para produzir um número médio de elementos, buscando um equilíbrio entre a diversidade de conteúdo e o tempo de processamento, evitando a escassez de elementos, o que tornaria o jogo trivial, e o excesso de informações, que poderia exceder a janela de contexto dos modelos menores. Os parâmetros definidos para cada execução foram:

- **Níveis da Masmorra:** 6 níveis de profundidade.
- **Inimigos:** 20 variações de inimigos.
- **Armas:** 30 tipos de armas.

5.1.1 Modelos e Cenários de Teste

Foram selecionados três LLMs para a tarefa de geração, ordenados de forma decrescente em relação à quantidade total de parâmetros, visando comparar modelos robustos contra modelos otimizados para hardware com recursos limitados:

1. **meta-llama/llama-4-maverick-17b-128e-instruct:** Modelo de maior escala baseado em arquitetura *Mixture of Experts* (MoE), contendo 400B de parâmetros totais (com 17B de parâmetros ativos durante a inferência), focado em alta aderência a instruções.
2. **openai/gpt-oss-120b:** Modelo denso de grande porte (117B parâmetros, 5.1B ativos), utilizado como referência de alta capacidade.
3. **openai/gpt-oss-20b:** Modelo compacto (21B parâmetros, 3.6B ativos), projetado para eficiência, sendo o candidato ideal para validação de inferência em hardware local.

Para a etapa de avaliação (*LLM-based Evaluation*), utilizou-se o modelo **Gemini 3 Pro** da Google como "juiz", devido à sua alta capacidade de raciocínio e janela de contexto

estendida. Para a métrica de reconstrução semântica, empregou-se o modelo de *embeddings* text-embedding-004. Todos os modelos foram configurados com temperatura de 0.4, um valor que balanceia criatividade e aderência às instruções, além de ser um padrão em trabalhos atuais na área como o de Nasir *et al.*. Os testes foram conduzidos sobre cinco *prompts* (P1 a P5) abrangendo temas distintos, comuns ao gênero *roguelike*:

- **P1:** *The Cursed Dwarven Forge* (Medieval/Fantasia).
- **P2:** *The Derelict Starship* (Ficção Científica/Espaço).
- **P3:** *The Smuggler's Grotto* (Pirata/Náutico).
- **P4:** *Neon Skyline Penthouse* (Cyberpunk/Urbano).
- **P5:** *The Living Hive* (Alien/Bio-Horror).

5.2 Análise Quantitativa

A avaliação quantitativa focou em dois aspectos principais: a coerência do conteúdo gerado e a fidelidade semântica (reconstrução).

A Tabela 1 apresenta os resultados da métrica de Coerência (escala 0-100), avaliada pelo Gemini 3 Pro. Observa-se que o modelo gpt-oss-120b obteve o melhor desempenho médio (93,6). O modelo llama-4-maverick-17b, apesar de sua arquitetura massiva (400B), apresentou uma média de 89 pontos. Importante notar o desempenho do modelo compacto gpt-oss-20b, que atingiu média de 87,8 pontos. A proximidade dos resultados do modelo menor em relação aos gigantes valida a hipótese de que modelos compactos são viáveis para a geração procedural local, mantendo um nível aceitável de coerência.

Tabela 1 – Resultados da Métrica de Coerência (0-100)

Modelo	P1	P2	P3	P4	P5	Média
llama-4-maverick-17b	95	65	92	95	98	89,0
openai/gpt-oss-120b	82	95	98	95	98	93,6
openai/gpt-oss-20b	95	68	95	85	96	87,8

Fonte: Elaborado pelo autor

A Tabela 2 exibe a Métrica de Reconstrução Semântica (Similaridade de Cosseno), que mede o quão bem os ativos gerados representam o tema original. Todos os modelos apresentaram alta fidelidade, com valores médios próximos a 0,89. O gpt-oss-120b obteve o melhor resultado individual no P2 (0,921), indicando uma excelente capacidade de capturar as nuances do tema de ficção científica, enquanto o modelo compacto gpt-oss-20b demonstrou

ser altamente competitivo, com média de 0,888.

Tabela 2 – Resultados da Métrica de Reconstrução (Similaridade de Cosseno)

Modelo	P1	P2	P3	P4	P5	Média
llama-4-maverick-17b	0,880	0,883	0,890	0,882	0,905	0,888
openai/gpt-oss-120b	0,877	0,921	0,908	0,895	0,867	0,894
openai/gpt-oss-20b	0,882	0,913	0,865	0,888	0,895	0,888

Fonte: Elaborado pelo autor

5.3 Análise Qualitativa

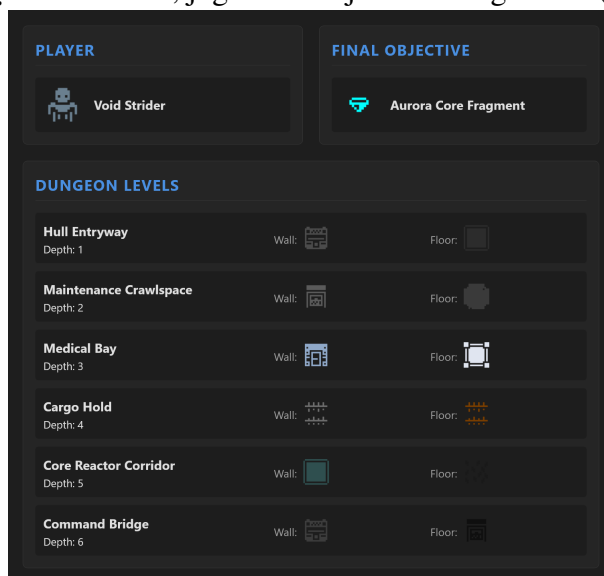
Para compreender a qualidade subjetiva e a coesão visual dos *assets* gerados, selecionaram-se dois casos de destaque para análise aprofundada: o modelo **openai/gpt-oss-120b** no tema *Sci-Fi* (P2), que obteve a melhor reconstrução, e o modelo **meta-llama/llama-4-maverick-17b** no tema *Alien/Bio-Horror* (P5), que demonstrou alta especificidade em suas instruções.

5.3.1 Caso de Estudo 1: *The Derelict Starship (GPT-120b)*

Neste cenário, o modelo gerou um pacote denominado "*The Derelict - The Void-Strider Ark*". A descrição textual do ambiente evoca uma atmosfera de horror industrial espacial, mencionando "painéis de aço reforçado", "dutos expostos" e "vazamentos de vácuo". A Figura 14 ilustra os níveis, o jogador e o objetivo final gerados. O jogador, "*Void Strider*", é descrito como um sucateiro em um traje exoesqueleto. O sistema de RAG recuperou corretamente um tile de personagem humanoide para representá-lo. Os níveis progridem logicamente de "*Hull Entryway*" (Entrada do Casco) até "*Command Bridge*" (Ponte de Comando), com texturas que refletem ambientes metálicos e tecnológicos. O objetivo final, "*Aurora Core Fragment*", está alinhado com a narrativa de uma IA corrompida.

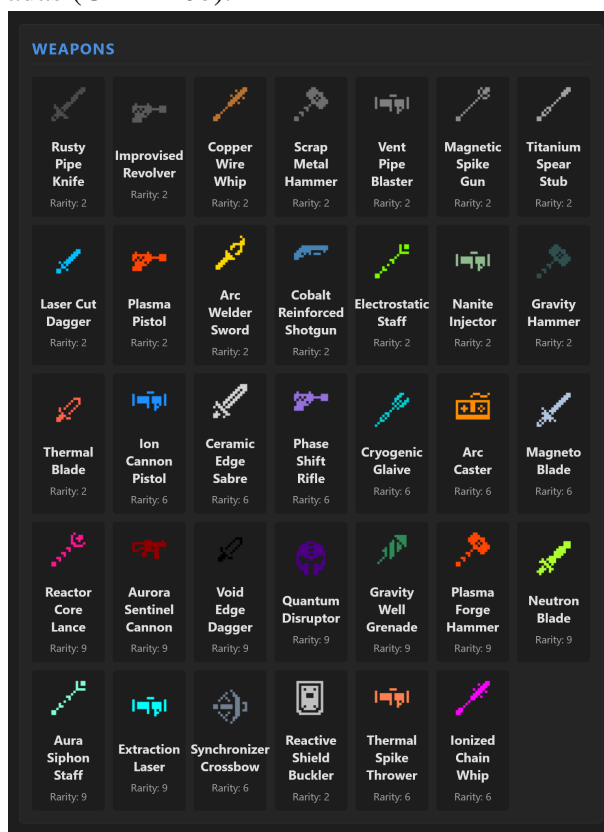
A coesão temática se estende aos inimigos (Figura 16) e armas (Figura 15). Os inimigos incluem "*Scout Spider*", "*Circuit Razor*" e o chefe "*Core Sentinel*", todos descritos como drones ou construtos mecânicos. As texturas recuperadas exibem tons de cinza, azul e vermelho, típicos de máquinas. As armas, como "*Plasma Pistol*" e "*Arc Welder Sword*", misturam tecnologia futurista com ferramentas industriais improvisadas, reforçando o tema de sobrevivência em uma nave abandonada.

Figura 14 – Visualização dos níveis, jogador e objetivo final gerados (GPT-120b).



Fonte: Elaborado pelo autor

Figura 15 – Armas geradas (GPT-120b).

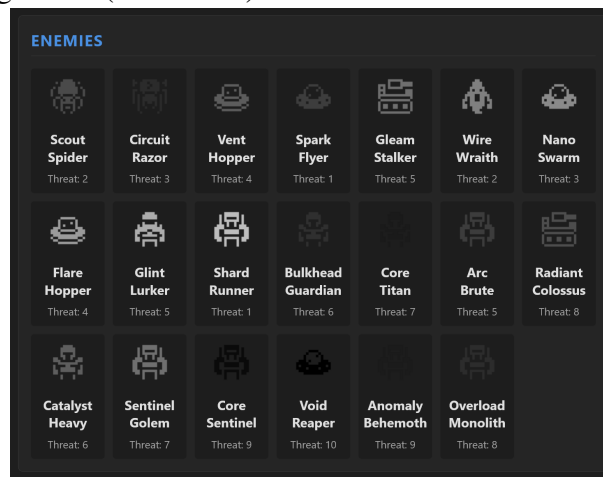


Fonte: Elaborado pelo autor

5.3.2 Caso de Estudo 2: The Living Hive (Llama-17b)

O modelo Llama-17b demonstrou excepcional capacidade de adaptação ao tema de horror biológico no cenário P5, gerando o pacote "Gutrix". Tirando proveito de sua vasta quantidade de parâmetros totais (400B) e treinamento em instruções, o modelo construiu uma

Figura 16 – Inimigos gerados (GPT-120b).

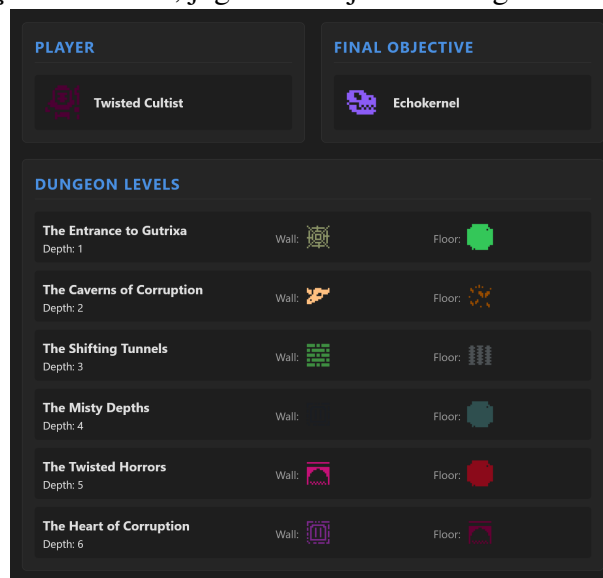


Fonte: Elaborado pelo autor

narrativa que descreve o interior de um organismo colossal, rejeitando linhas retas em favor de "tecidos musculares" e "câmaras esféricas".

A Figura 17 apresenta os resultados estruturais. O jogador é um "*Twisted Cultist*" (Cultista Distorcido), cuja história de fundo envolve uma seita que adora a criatura planetária. Os níveis, como "*The Caverns of Corruption*" e "*The Heart of Corruption*", utilizam texturas orgânicas e cores vibrantes (roxo, verde, rosa), contrastando fortemente com o exemplo anterior.

Figura 17 – Visualização dos níveis, jogador e objetivo final gerados (Llama-17b).

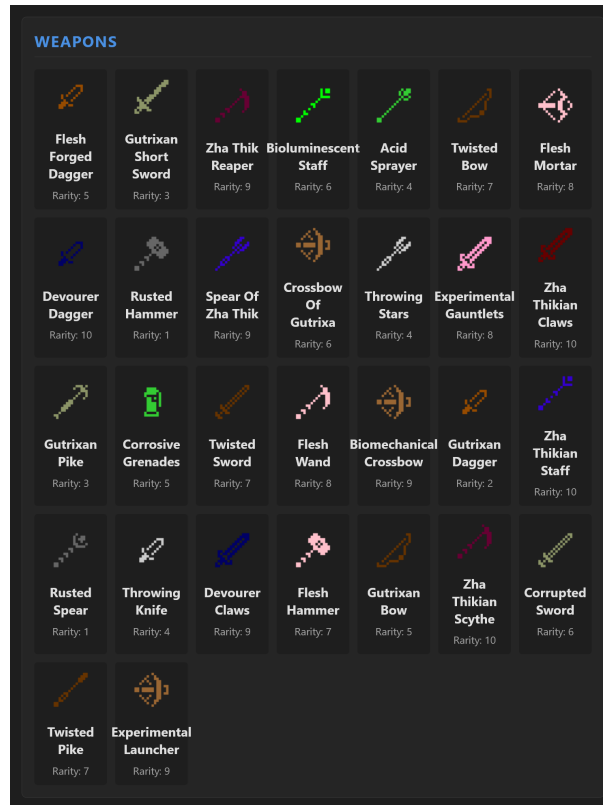


Fonte: Elaborado pelo autor

Os inimigos gerados (Figura 19) reforçam o tema grotesco, com criaturas como "*Gutrixa Groaner*", "*Flesh Borer*" e "*Zhathik Titan*". As descrições enfatizam características insectoides e carnosas. As armas (Figura 18), como "*Flesh Forged Dagger*" e "*Acid Sprayer*",

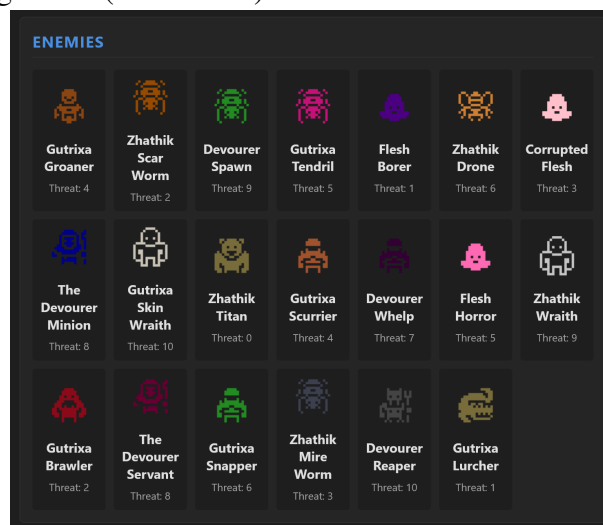
utilizam materiais orgânicos (ossos, tendões) em vez de metal, demonstrando que o modelo compreendeu a restrição de materiais do mundo. O objetivo final, "*Echokernel*", é descrito como o coração corrompido da entidade, fechando o ciclo narrativo.

Figura 18 – Armas geradas (Llama-17b).



Fonte: Elaborado pelo autor

Figura 19 – Inimigos gerados (Llama-17b).



Fonte: Elaborado pelo autor

A comparação qualitativa indica que, embora o GPT-120b tenda a gerar descrições

ligeiramente mais detalhadas, o modelo Llama-17b foi capaz de capturar a essência estilística do tema com precisão surpreendente, selecionando paletas de cores e terminologias adequadas (ex: "*flesh*", "*sinew*", "*chitinous*") que guiaram o sistema de RAG para texturas visualmente distintas do cenário de ficção científica.

5.4 Jogo Roguelike

Além da geração dos ativos textuais e visuais, uma etapa fundamental deste trabalho foi a validação destes elementos em um ambiente de jogo funcional. Utilizando a *game engine* *Godot* em sua versão 4.5.1, foi desenvolvido um jogo *Roguelike* capaz de processar um *Asset-Bundle* em formato JSON e transformá-lo em um mapa jogável. Esta seção detalha os sistemas básicos implementados que compõem a experiência de jogo, desde a interface do usuário até os algoritmos de visibilidade e distribuição procedural.

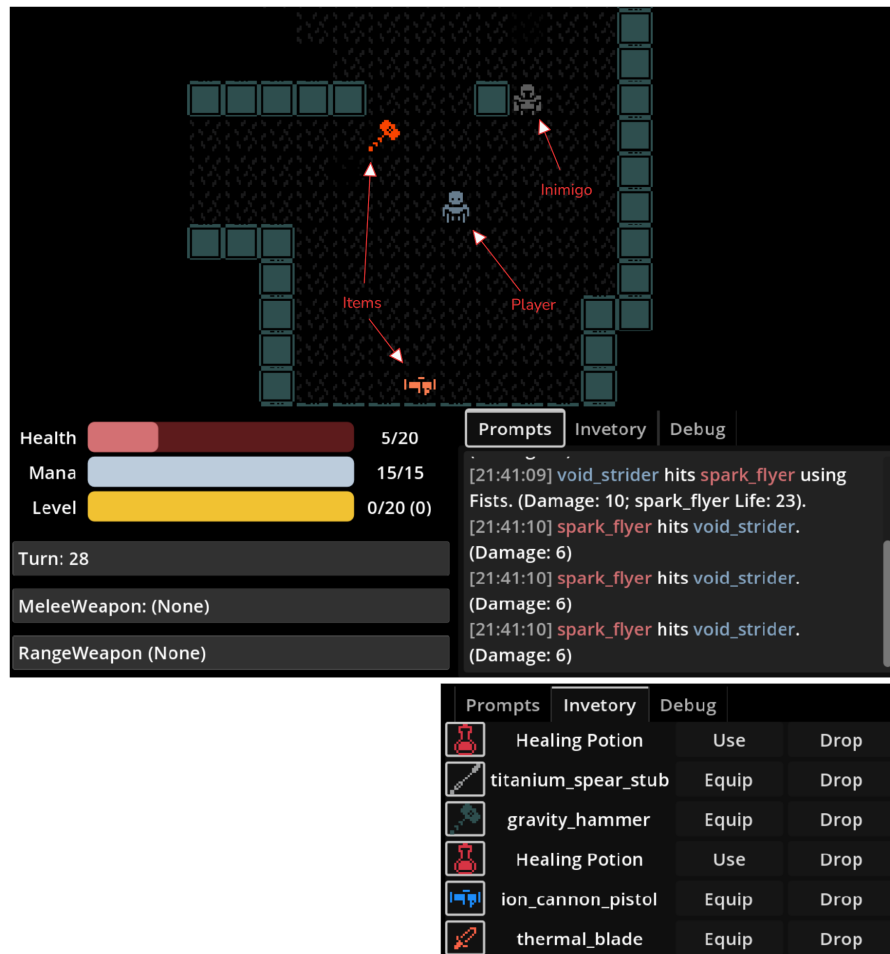
5.4.1 Interface e Ciclo de Gameplay

A Figura 20 apresenta uma visão geral da interface principal do jogo durante uma partida. A implementação segue estritamente as convenções do gênero. O núcleo do jogo opera através de um sistema de turnos, exibido no canto esquerdo da tela (abaixo das barras de status). Os turnos são incrementados apenas após o jogador realizar uma ação significativa, como mover-se, atacar ou utilizar um item. Esta mecânica é crucial para o aspecto tático do gênero, permitindo que o jogador tenha tempo ilimitado para planejar suas ações, decidir entre atacar um inimigo próximo, utilizar uma poção de cura ou recuar estrategicamente, antes que o ambiente reaja.

No canto inferior esquerdo, o *Heads-Up Display* (HUD) fornece informações sobre o estado do personagem através de três barras de progresso:

- **Barra Vermelha (Vida):** Representa a saúde do jogador. Se esta barra chegar a zero, o personagem morre, resultando na perda de todos os itens e do progresso atual, forçando o reinício em um novo mapa gerado aleatoriamente.
- **Barra Azul (Mana):** Indica o recurso necessário para a utilização de armas de longo alcance.
- **Barra Amarela (Experiência):** Monitora o progresso do jogador. Ao atingir o máximo, o personagem sobe de nível, restaurando e incrementando seus atributos máximos de vida e

Figura 20 – Visão geral da interface do jogo, destacando HUD, logs e inventário.



Fonte: Elaborado pelo autor

mana.

Abaixo das barras, são exibidos os nomes da arma corpo a corpo (*melee*) e da arma de ataque à distância atualmente equipadas. No lado direito da tela, uma barra de menus permite ao jogador alternar entre duas abas principais: "*Prompts*" e "*Inventory*". A aba de *Prompts* atua como um registro de eventos (*combat log*), exibindo mensagens informativas como dano causado ou recebido e descrições de itens encontrados no chão. A aba de *Inventory* permite o gerenciamento de recursos. O jogador pode equipar armas coletadas ou consumir itens como poções de cura e mana. A cena é composta ainda pela renderização do mapa, no qual é possível observar o jogador centralizado, itens espalhados pelo chão e inimigos patrulhando a área.

5.4.2 Sistema de Campo de Visão (FOV)

A exploração e a descoberta são pilares dos *roguelikes*. Para suportar isso, foi implementado um sistema de Campo de Visão, ilustrado na Figura 21.

Figura 21 – Sistema de Campo de Visão: distinção entre áreas visíveis, exploradas e não exploradas.



Fonte: Elaborado pelo autor

O sistema divide a renderização do mapa em três estados distintos:

1. **Tiles Visíveis:** A área iluminada ao redor do jogador, onde entidades (inimigos e itens) são renderizadas em tempo real.
2. **Tiles Explorados (Fora do FOV):** Áreas que o jogador já visitou, mas que não estão em sua linha de visão direta no momento. Estes *tiles* permanecem visíveis, porém escurecidos e estáticos (não mostram movimentação de inimigos), servindo como memória espacial para o jogador.
3. **Tiles Não Explorados:** Áreas completamente ocultas (preto), que representam o desconhecido.

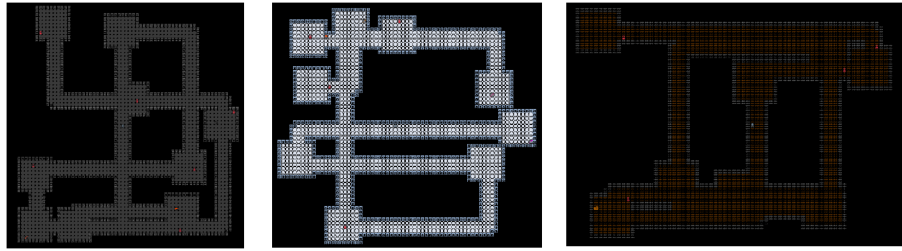
Este mecanismo garante que o jogador tenha apenas uma noção parcial da estrutura da masmorra, preservando a sensação de surpresa e perigo ao adentrar novas salas.

5.4.3 Geração Procedural e Progressão de Dificuldade

A estrutura física das masmorras é criada utilizando o Algoritmo 1 descrito na metodologia. A Figura 22 demonstra três gerações distintas de níveis da masmorra, evidenciando a variabilidade topológica obtida.

Cada mapa é composto por múltiplas salas retangulares interconectadas por corredores. Após a definição da geometria, o sistema popula as salas com itens e inimigos baseando-se nos metadados de *rarity*, *threat* e *weight* definidos pelo LLM. O algoritmo de povoamento utiliza uma curva de probabilidade ajustada pela profundidade do nível: itens raros e inimigos com alto nível de ameaça possuem chances mínimas de aparecer nos primeiros níveis, mas tornam-se progressivamente mais comuns conforme o jogador desce para profundidades maiores. Esta

Figura 22 – Três exemplos de níveis gerados proceduralmente com diferentes configurações espaciais.



Fonte: Elaborado pelo autor

abordagem assegura não apenas a integridade estrutural do mapa, mas também um senso tangível de progressão de dificuldade e recompensa.

6 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho investigou o potencial e a viabilidade da utilização de LLMs no contexto da PCG para jogos do gênero *roguelike*. A pesquisa partiu da identificação de uma lacuna significativa na literatura e nas capacidades técnicas atuais: embora os LLMs demonstrem proficiência semântica e criativa, eles possuem limitações intrínsecas no raciocínio espacial e geométrico, o que dificulta a geração direta de topologias de mapas complexos e funcionais.

Para solucionar este problema, foi proposta e implementada uma arquitetura híbrida baseada no desacoplamento de responsabilidades. O sistema delegou a construção estrutural e topológica da masmorra a algoritmos de PCG tradicionais e determinísticos, garantindo a jogabilidade e a conectividade das salas. Simultaneamente, atribuiu-se ao LLM a função de "agente criativo", responsável pela geração semântica de ativos, incluindo a definição de temas, narrativas, inimigos e itens. Esta abordagem permitiu contornar as deficiências espaciais dos modelos de linguagem, aproveitando sua capacidade de interpretar intenções textuais e enriquecer o contexto do jogo.

Um componente central para o sucesso desta implementação foi a utilização da técnica de RAG aliada ao uso de *Structured Outputs*. A segmentação do banco vetorial em três categorias distintas (*Entities*, *Items* e *Environments*) provou-se uma estratégia eficaz para mitigar alucinações visuais, impedindo, por exemplo, a seleção de texturas de arquitetura para representar personagens. A curadoria e anotação manual deste *dataset*, composto por 489 pares de descrições e imagens, constitui uma contribuição técnica relevante deste trabalho, estabelecendo uma base sólida para a associação entre linguagem natural e ativos de *pixel art*.

A validação do sistema, realizada através de uma abordagem quantitativa e qualitativa, corroborou a eficácia do *pipeline* proposto. A aplicação da métrica de Reconstrução Semântica, baseada na similaridade de cosseno, demonstrou que o sistema é capaz de preservar a intenção original do usuário ao traduzir descrições textuais em objetos de jogo. Mais notavelmente, os testes comparativos entre diferentes modelos revelaram que LLMs compactos e executados localmente, como o `openai/gpt-oss-20b`, apresentam desempenho pareável a modelos de maior porte e complexidade como o `gpt-oss-120b` e o `llama-4-maverick`, confirmando a viabilidade técnica de integrar esta tecnologia em hardware de consumo, sem dependência mandatória de conexões com a nuvem.

Em suma, este trabalho demonstra que a integração entre algoritmos clássicos de PCG e a moderna Inteligência Artificial Generativa não apenas é possível, como também

oferece um caminho promissor para o desenvolvimento de jogos. O protótipo desenvolvido exemplifica como a barreira técnica da criação de conteúdo pode ser reduzida, permitindo que desenvolvedores e jogadores utilizem linguagem natural para moldar experiências de jogo únicas e coerentes.

6.1 Limitações e Trabalhos Futuros

Apesar dos resultados satisfatórios obtidos com a arquitetura proposta, o sistema apresenta limitações que abrem caminhos para futuras investigações e aprimoramentos. A restrição mais significativa reside na dependência de um banco vetorial estático para a representação visual. O mecanismo de RAG, embora eficiente na recuperação, está limitado ao universo de texturas pré-existentes no *dataset*. Em cenários nos quais a descrição textual do usuário solicita um objeto específico não catalogado (por exemplo, uma "foice"), o sistema seleciona a aproximação semântica mais próxima disponível (como uma "espada"), o que pode gerar inconsistências visuais e reduzir a liberdade criativa.

Adicionalmente, o escopo atual de geração restringe-se a entidades isoladas. O sistema instancia objetos individuais, mas ainda não possui a capacidade de orquestrar a geração de estruturas compostas ou arranjos arquitetônicos complexos, como monumentos, cemitérios ou acampamentos, que exigiriam um entendimento mais refinado de composição de cena por parte do LLM. Diante destes desafios, sugerem-se as seguintes direções para trabalhos futuros:

1. **Integração com IA Generativa de Imagens:** A substituição ou complementação do banco vetorial fixo por modelos de difusão latente (como *Stable Diffusion*) gerando texturas em tempo real. Isso eliminaria a limitação do acervo de ativos, permitindo que a representação visual seja tão dinâmica e ilimitada quanto as descrições textuais geradas pelo LLM.
2. **Geração de Estruturas Complexas:** Expandir a lógica do *Gerador de Assets* para que o LLM possa definir não apenas objetos únicos, mas conjuntos de elementos pré-fabricados (*prefabs*). Isso permitiria povoar o mapa com micro-cenários semanticamente coesos, aumentando a profundidade da ambientação.
3. **Aplicação em Novos Gêneros:** Adaptar a metodologia para outros gêneros de jogos, como RPGs, Plataforma ou *Puzzles*. No contexto de RPGs, por exemplo, o sistema poderia ser expandido para gerar NPCs com árvores de diálogos complexas, missões personalizadas e histórias de fundo que reagem dinamicamente ao tema escolhido pelo jogador, explorando ainda mais a capacidade narrativa dos modelos de linguagem.

REFERÊNCIAS

- ALBERT, M. What are roguelites? a deep dive. abr. 2023. Disponível em: <https://hackernoon.com/what-are-roguelites-a-deep-dive>. Acesso em: 27 jan. 2026.
- AUXTERO, A. L. S. **Game Environment Design Creator Using Artificial Intelligence Procedural Generation**. 1–97 p. Master Thesis (Computer Science and Engineering) – NOVA University, Lisbon, Portugal, 2023. Disponível em: <https://run.unl.pt/entities/publication/72b2900b-031c-4263-a7cd-d4759bc79e5d>. Acesso em: 27 jan. 2026.
- BOTTA, C. L. B. **Oh No, I'm Dead! ... Again**: Death as a crossroad where different pathways in the study of video games converge. 1–96 p. Master Thesis (Digital Media) – University of Applied Sciences, Bremerhaven, Germany, 2011. Disponível em: https://www.academia.edu/44443054/Oh_No_Im_Dead_Again_Death_as_a_Crossroad_Where_Different_Pathways_in_the_Study_of_Video_Games_Converge. Acesso em: 27 jan. 2026.
- BROWN, P. F.; DESOUZA, P. V.; MERCER, R. L.; PIETRA, V. J. D.; LAI, J. C. Class-based n-gram models of natural language. **Computational Linguistics**, Cambridge, v. 18, n. 4, p. 467–479, 1992. Disponível em: <https://dl.acm.org/doi/10.5555/176313.176316>. Acesso em: 28 jan. 2026.
- BYCER, J. The roguelike debate: Roguelikes vs roguelites. nov. 2019. Disponível em: <https://www.gamedeveloper.com/design/the-roguelike-debate----roguelikes-vs-roguelites>. Acesso em: 27 jan. 2026.
- CARTLIDGE, J. **Genre, Prototype Theory and the Berlin Interpretation of Roguelikes**. 2024. Disponível em: <https://gamestudies.org/2403/articles/cartlidge>. Acesso em: 27 jan. 2026.
- DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. Bert: pre-training of deep bidirectional transformers for language understanding. In: **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)**. Minneapolis: Association for Computational Linguistics, 2019. p. 4171–4186. Disponível em: <https://aclanthology.org/N19-1423/>. Acesso em: 28 jan. 2026.
- EHRIG, H.; PFENDER, M.; SCHNEIDER, H. J. Graph-grammars: An algebraic approach. In: IEEE, 14., 1973, Iowa City. **Proceedings of 14th Annual Symposium on Switching and Automata Theory (swat 1973)**. Iowa City, Iowa: Institute of Electrical and Electronics Engineers, 1973. p. 167–180. Disponível em: <https://ieeexplore.ieee.org/document/4569741>. Acesso em: 27 jan. 2026.
- EPYX, INC. **Rogue Instruction Manual**. Sunnyvale, CA, 1985. Versão para IBM PC e PCjr. Disponível em: <https://britzl.github.io/roguearchive/files/misc/EpyxRogueDOSManual.pdf>. Acesso em: 27 jan. 2026.
- GALLOTTA, R.; LIAPIS, A.; YANNAKAKIS, G. Consistent game content creation via function calling for large language models. In: **Proceedings of 2024 IEEE Conference on Games (CoG)**. Milan, Italy: [S. n.], 2024. p. 1–4. Disponível em: <https://ieeexplore.ieee.org/document/10645599>. Acesso em: 27 jan. 2026.

GALLOTTA, R.; TODD, G.; ZAMMIT, M.; EARLE, S.; LIAPIS, A.; TOGELIUS, J.; YANNAKAKIS, G. N. Large language models and games: A survey and roadmap. **IEEE Transactions on Games**, New York, p. 1–18, 2024. Disponível em: <https://arxiv.org/abs/2402.18659>. Acesso em: 27 jan. 2026.

HADI, M. U.; QURESHI, R.; SHAH, A.; IRFAN, M.; ZAFAR, A.; SHAIKH, M. B.; AKHTAR, N.; WU, J.; MIRJALILI, S. *et al.* **A survey on large language models**: Applications, challenges, limitations, and practical usage. 2023. Disponível em: <https://arxiv.org/abs/2303.18223>. Acesso em: 27 jan. 2026.

HARRIS, J. **Exploring roguelike games**. 1. ed. [S. l.]: CRC Press, 2020. 556 p.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, Cambridge, v. 9, n. 8, p. 1735–1780, 1997. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>. Acesso em: 28 jan. 2026.

HUANG, S. **Word2Minecraft**: Generating 3d game levels through large language models. 2025. Disponível em: <https://arxiv.org/abs/2307.06435>. Acesso em: 27 jan. 2026.

KOESNAEDI, A.; ISTIONO, W. Implementation drunkard’s walk algorithm to generate random level in roguelike games. **International Journal of Multidisciplinary Research and Publications**, v. 5, n. 2, p. 97–103, 2022. Disponível em: <https://ijmrmap.com/wp-content/uploads/2022/07/IJMRAP-V5N2P27Y22.pdf>. Acesso em: 27 jan. 2026.

LAURINDO, E. K.; OTI, K. H.; BROCATTO, M. d. A.; BELCHIOR, V. **Relatório técnico de desenvolvimento do jogo digital Thorn Ascent**. 1–128 p. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Jogos Digitais) – Faculdade de Tecnologia de Americana “Ministro Ralph BIASI”, São Paulo, 2024. Disponível em: https://ric.cps.sp.gov.br/handle/123456789/21637?locale=pt_BR. Acesso em: 27 jan. 2026.

LAZARIDIS, L.; FRAGULIS, G. F. Creating a newer and improved procedural content generation (pcg) algorithm with minimal human intervention for computer gaming development. **Computers**, MDPI, v. 13, n. 11, p. 304, 2024. Disponível em: <https://www.mdpi.com/2073-431X/13/11/304>. Acesso em: 27 jan. 2026.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, New York, v. 86, n. 11, p. 2278–2324, 1998. Disponível em: <https://doi.org/10.1109/5.726791>. Acesso em: 28 jan. 2026.

LEVIN, D. A.; PERES, Y. **Markov chains and mixing times**. 2. ed. [S. l.]: American Mathematical Soc., 2017. v. 107. 464 p.

LIN, T.; WANG, Y.; LIU, X.; QIU, X. **A Survey of Transformers**. 2021. Disponível em: <https://arxiv.org/abs/2106.04554>. Acesso em: 27 jan. 2026.

LINDENMAYER, A. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. **Journal of Theoretical Biology**, v. 18, n. 3, p. 280–299, 1968. Disponível em: <https://www.sciencedirect.com/science/article/pii/0022519368900799>. Acesso em: 27 jan. 2026.

LIU, Y.; HE, H.; HAN, T.; ZHANG, X.; LIU, M.; TIAN, J.; ZHANG, Y.; WANG, J.; GAO, X.; ZHONG, T. *et al.* **Understanding llms**: A comprehensive overview from training to inference. 2024. Disponível em: <https://arxiv.org/abs/2401.02038>. Acesso em: 27 jan. 2026.

LOH, W.-Y. Classification and regression trees. **Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery**, Hoboken, v. 1, n. 1, p. 14–23, 2011. Disponível em: <https://doi.org/10.1002/widm.8>. Acesso em: 28 jan. 2026.

MALEKI, M. F.; ZHAO, R. Procedural content generation in games: a survey with insights on emerging llm integration. In: **Proceedings of the Twentieth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment**. University of Kentucky in Lexington, Kentucky, USA: AAAI Press, 2024. v. 20, n. 17, p. 167–178. Disponível em: <https://doi.org/10.1609/aiide.v20i1.31877>. Acesso em: 27 jan. 2026.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The Bulletin of Mathematical Biophysics**, Chicago, v. 5, n. 4, p. 115–133, 1943. Disponível em: <https://doi.org/10.1007/BF02478259>. Acesso em: 28 jan. 2026.

NASIR, M. U.; JAMES, S.; TOGELIUS, J. **Word2world**: Generating stories and worlds through large language models. 2024. Disponível em: <https://arxiv.org/abs/2405.06686>. Acesso em: 27 jan. 2026.

NAVEED, H.; KHAN, A. U.; QIU, S.; SAQIB, M.; ANWAR, S.; USMAN, M.; AKHTAR, N.; BARNES, N.; MIAN, A. **A comprehensive overview of large language models**. 2024. Disponível em: <https://arxiv.org/abs/2307.06435>. Acesso em: 27 jan. 2026.

PEARSON, K. The problem of the random walk. **Nature**, London, v. 72, n. 1865, p. 294, 1905. Disponível em: <https://doi.org/10.1038/072294b0>. Acesso em: 28 jan. 2026.

PERLIN, K. An image synthesizer. In: **Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques**. New York, NY, USA: Association for Computing Machinery, 1985. p. 287–296. Disponível em: <https://doi.org/10.1145/325334.325247>. Acesso em: 27 jan. 2026.

PERLIN, K. Noise hardware. In: OLANO, M. (Ed.). **Real-Time Shading SIGGRAPH Course Notes**. ACM SIGGRAPH, 2001. Course Notes. Disponível em: <https://userpages.cs.umbc.edu/olano/s2002c36/ch02.pdf>. Acesso em: 27 jan. 2026.

RADFORD, A.; NARASIMHAN, K.; SALIMANS, T.; SUTSKEVER, I. **Improving language understanding by generative pre-training**. San Francisco, 2018. Disponível em: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf. Acesso em: 28 jan. 2026.

ROLIM, F. M. **LLMs e IA generativa em jogos**. 1–68 p. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2023. Disponível em: <https://pantheon.ufrj.br/handle/11422/22275>. Acesso em: 27 jan. 2026.

SALMELA, T. **Game development using the open-source Godot Game Engine**. 1–67 p. Bachelor's Thesis (Media and Arts) – Tampere University of Applied Sciences, Tampere, Finlândia, 2022. Disponível em: <https://www.theseus.fi/handle/10024/746943>. Acesso em: 27 jan. 2026.

SEPÄNMAA, T. **Procedural Level Generation in 2D Roguelite Games**. 1–36 p. Bachelor's Thesis (Business Information Systems) – Tampere University of Applied Sciences, Tampere, Finlândia, 2024. Disponível em: https://www.theseus.fi/bitstream/handle/10024/873015/Sepanmaa_Tomi.pdf?sequence=2. Acesso em: 27 jan. 2026.

SHORT, T.; ADAMS, T. **Procedural generation in game design**. 1. ed. [S. l.]: A K Peters/CRC Press, 2017. 336 p.

SHORTEN, C.; PIERSE, C.; SMITH, T. B.; CARDENAS, E.; SHARMA, A.; TRENGROVE, J.; LUIJT, B. van. **Structuredrag**: Json response formatting with large language models. 2024. Disponível em: <https://arxiv.org/abs/2408.11061>. Acesso em: 27 jan. 2026.

SMITH, G.; GAN, E.; OTHENIN-GIRARD, A.; WHITEHEAD, J. Pcg-based game design: enabling new play experiences through procedural content generation. In: **Proceedings of the 2nd International Workshop on Procedural Content Generation in Games**. New York, NY, USA: Association for Computing Machinery, 2011. (PCGames '11), p. 1–4. Disponível em: <https://doi.org/10.1145/2000919.2000926>. Acesso em: 27 jan. 2026.

SOUZA, R.; JUNIOR, B.; FOSS, L.; CAVALHEIRO, G.; CAVALHEIRO, S. Geração procedural de mapas dungeon crawl baseada em gramática de grafos para uso em jogos roguelike. In: **Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho**. Porto Alegre, RS, Brasil: SBC, 2019. p. 193–203. Disponível em: <https://sol.sbc.org.br/index.php/sscad/article/view/8668>. Acesso em: 27 jan. 2026.

STINY, G.; GIPS, J. Shape grammars and the generative specification of painting and sculpture. In: FREIMAN, C. V. (Ed.). **Proceedings of Information Processing 71**. Amsterdam: North-Holland, 1972. p. 1460–1465. Disponível em: https://www.researchgate.net/publication/221329330_Shape_Grammars_and_the_Generative_Specification_of_Painting_and_Sculpture. Acesso em: 27 jan. 2026.

SUDHAKARAN, S.; GONZÁLEZ-DUQUE, M.; FREIBERGER, M.; GLANOIS, C.; NAJARRO, E.; RISI, S. MarioGPT: open-ended text2level generation through large language models. In: **Proceedings of the 37th International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2023. p. 54213–54227. Disponível em: <https://dl.acm.org/doi/10.5555/3666122.3668483>. Acesso em: 27 jan. 2026.

TOGELIUS, J.; KASTBJERG, E.; SCHEDL, D.; YANNAKAKIS, G. N. What is procedural content generation? mario on the borderline. In: **Proceedings of the 2nd International Workshop on Procedural Content Generation in Games**. New York, NY, USA: Association for Computing Machinery, 2011. (PCGames '11), p. 1–6. Disponível em: <https://doi.org/10.1145/2000919.2000922>. Acesso em: 27 jan. 2026.

TOGELIUS, J.; YANNAKAKIS, G. N.; STANLEY, K. O.; BROWNE, C. Search-based procedural content generation: a taxonomy and survey. **IEEE Transactions on Computational Intelligence and AI in Games**, New York, v. 3, n. 3, p. 172–186, 2011. Disponível em: <https://doi.org/10.1109/TCIAIG.2011.2148116>. Acesso em: 28 jan. 2026.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. In: **Proceedings of the 31st International Conference on Neural Information Processing Systems**. Red Hook, NY, USA: Curran Associates Inc., 2017. p. 6000–6010. Disponível em: <https://dl.acm.org/doi/10.5555/3295222.3295349>. Acesso em: 27 jan. 2026.

WIJMAN, T. **Newzoo's year in review: the 2023 global games market in numbers.** 2023. Disponível em: <https://newzoo.com/resources/blog/video-games-in-2023-the-year-in-numbers>. Acesso em: 27 jan. 2026.

YAN, H.; HU, X.; WAN, X.; HUANG, C.; ZOU, K.; XU, S. **Inherent limitations of LLMs regarding spatial information.** 2023. Disponível em: <https://arxiv.org/abs/2312.03042>. Acesso em: 27 jan. 2026.

APÊNDICE A – EXEMPLO DE ASSET BUNDLE EM JSON

Código-fonte 1 – Exemplo de saída JSON

```
1 {
2   "player": {
3     "back_history": "...",
4     "tile_with_texture": {
5       "name": "...",
6       "description": "...",
7       "color": "...",
8       "texture": {
9         "tileset_position": {
10           "x": 0,
11           "y": 0
12         }
13       }
14     }
15  },
16  "final_objective": {
17    "back_history": "...",
18    "tile_with_texture": {
19      ...
20    }
21  },
22  "dungeon_levels": {
23    "items": [
24      {
25        "description": "...",
26        "name": "...",
27        "depth": 0,
28        "wall_tile_with_texture": {
29          ...
30        },
31        "floor_tile_with_texture": {
32          ...
33        }
34      }
35    ]
36  },
37  "enemies": {
38    "items": [
39      {
40        "weight": 0,
41        "thread": 0,
42        "tile_with_texture": {
```

```
43         ...
44     }
45 }
46 ]
47 },
48 "weapons": {
49     "items": [
50         {
51             "rarity": 0,
52             "weight": 0,
53             "mana_cost": 0,
54             "weapon_type": "...",
55             "tile_with_texture": {
56                 ...
57             }
58         }
59     ]
60 }
61 }
```