**UNIVERSIDADE FEDERAL DE JUIZ DE FORA**
**FACULDADE DE ENGENHARIA**
**BACHARELADO EM ENGENHARIA COMPUTACIONAL**

**Gabriel Rodrigues Toledo**

# Geração procedural de biomas em mapas de jogos bidimensionais

Juiz de Fora
2026

**Gabriel Rodrigues Toledo**

# Geração procedural de biomas em mapas de jogos bidimensionais

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Bacharel em Engenharia Computacional.

Orientador: Prof. Dr. Marcelo Caniato Renhe

Juiz de Fora

2026

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Rodrigues Toledo, Gabriel.
Geração procedural de biomas em mapas de jogos bidimensionais / Gabriel Rodrigues Toledo. – 2026.
65 f. : il.

Orientador: Marcelo Caniato Renhe
Trabalho de Conclusão de Curso (Graduação) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia. Bacharelado em Engenharia Computacional, 2026.

1. Geração Procedural. 2. Mapas Bidimensionais. 3. Terraria. I. Caniato Renhe, Marcelo, orient. II. Título.

**Gabriel Rodrigues Toledo**

**Geração procedural de biomas em mapas de jogos bidimensionais**

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Bacharel em Engenharia Computacional.

Aprovada em 23 de janeiro de 2026

BANCA EXAMINADORA

---

Prof. Dr. Marcelo Caniato Renhe - Orientador
Universidade Federal de Juiz de Fora

---

Prof. Dr. Igor de Oliveira Knop
Universidade Federal de Juiz de Fora

---

Prof. Dr. Luiz Maurilio da Silva Maciel
Universidade Federal de Juiz de Fora

## AGRADECIMENTOS

Agradeço primeiramente a todos os professores que fizeram parte da minha formação acadêmica, pelos conhecimentos compartilhados e pelos desafios propostos ao longo do curso. Um agradecimento muito especial ao meu orientador, Marcelo Caniato Renhe, pela sua paciência, pela orientação precisa e, acima de tudo, por não ter desistido de mim. Sem o seu apoio, a concretização deste trabalho não seria possível.

Aos meus pais, Maria Alice e Paulo Roberto, que são minhas maiores inspirações. Obrigado por serem meus maiores apoiadores, pelo amor incondicional e por tornarem possível que eu chegasse até aqui. Ao meu irmão, Pedro Paulo, que sempre esteve ao meu lado, pela amizade e companheirismo durante toda a vida. À Tatiane, por todo o apoio, carinho e também pelas cobranças necessárias, que me incentivaram a manter o foco e concluir esta etapa.

Estendo minha gratidão a toda a minha família, tios, tias, primos e primas, que sempre me incentivaram, torceram pelo meu sucesso e me deram suporte. Por fim, agradeço aos meus amigos e a todos aqueles que, direta ou indiretamente, apoiaram-me e contribuíram para a realização deste trabalho.

**RESUMO**

A indústria de jogos eletrônicos demanda cada vez mais a criação de mundos vastos e complexos, tornando a Geração Procedural de Conteúdo (GPC) uma ferramenta essencial para garantir variabilidade e eficiência no desenvolvimento. Este trabalho explora técnicas de GPC para a criação de mapas bidimensionais em jogos de plataforma e exploração (*sandbox*), utilizando o jogo Terraria como estudo de caso para validação dos mapas gerados. O objetivo principal é compreender como a combinação de múltiplos algoritmos de GPC pode ser usada para criar um mundo virtual composto por regiões com diferentes características. A metodologia adotada usou o ruído de Perlin para a definição da macro-topografia e dos biomas, autômatos celulares para a geração de cavernas, e o algoritmo de caminhada aleatória para a escavação de túneis e geração de masmorras. Os resultados obtidos mostram que a abordagem híbrida foi eficaz na criação de mapas coerentes, permitindo uma reprodução fiel das características das camadas de solo, cavernas, biomas e estruturas específicas do jogo de referência. A validação dos mapas gerados neste trabalho foi feita por meio de comparação visual direta com mapas obtidos a partir do jogo original.

Palavras-chave: geração procedural; geração de mapas; jogos sandbox.

**ABSTRACT**

The electronic game industry increasingly demands the creation of vast and complex worlds, rendering Procedural Content Generation (PCG) an essential tool for ensuring replayability and development efficiency. This work explores PCG techniques for the creation of two-dimensional maps in platform and exploration (sandbox) games, utilizing the game Terraria as a case study for topological and structural validation. The main objective was to develop a system capable of orchestrating multiple algorithms to reproduce the geological complexity of a virtual world. The adopted methodology combined Perlin Noise for macro-topography and biomes, Cellular Automata for simulating erosion in deep caves, and Random Walk algorithms for excavating tunnels and architectural structures, such as dungeons. The results demonstrated that the hybrid and stratified approach is effective in generating coherent maps, allowing for the faithful reproduction of soil layers, caverns, biomes and specific structures of the reference game. The final system yielded deterministic maps based on seeds, validated through direct visual comparison with the original generation of the case study.

Keywords: procedural generation; bidimensional maps; terraria; perlin noise; cellular automata; random walk.

## LISTA DE FIGURAS

Lista de Algoritmos

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| UFJF | Universidade Federal de Juiz de Fora |
| GPC | Geração Procedural de Conteúdo |
| PCG | *Procedural Content Generation* |
| GANs | *Generative adversarial networks* |
| CPU | *Central Processing Unit* |
| fBm | Movimento Browniano Fractal |

**SUMÁRIO**

# 1 INTRODUÇÃO

A indústria de jogos eletrônicos tem demandado cada vez mais a criação de mundos vastos e complexos para manter o interesse dos jogadores. No entanto, a criação manual desses conteúdos consome tempo e recursos significativos. Neste cenário, a Geração de Conteúdo Procedural (GPC) surge como uma solução, permitindo que algoritmos criem dados automaticamente, variando desde texturas simples até mundos inteiros (Rose e Bakaoukas, 2016).

A geração procedural é especialmente relevante em jogos do gênero *sandbox* e exploração, como Terraria e Minecraft, onde a variabilidade é um fator crítico. A capacidade de oferecer uma experiência única a cada nova sessão depende da criação de mapas que sejam não apenas aleatórios, mas que possuam coerência estrutural, biomas distintos e desafios de navegação interessantes (Togelius et al, 2011).

Este trabalho explora o uso de diferentes técnicas de geração procedural para a criação de mapas bidimensionais compostos por diferentes biomas. A ideia é combinar múltiplos algoritmos clássicos de GPC, como o ruído de Perlin para a topografia base, autômatos celulares para a formação de cavernas orgânicas e caminhada aleatória para a criação de estruturas complexas como masmorras e túneis.

## 1.1 JUSTIFICATIVA E MOTIVAÇÃO

A motivação para este trabalho reside no desafio técnico de equilibrar a geração de elementos aleatórios com o design de níveis intencional. Embora a geração aleatória seja interessante para variar a experiência do jogador, a aleatoriedade pura não é capaz de produzir resultados úteis. Um mapa de jogo jogável requer regras e padrões definidos. Compreender como diversos algoritmos podem ser orquestrados para criar ambientes coesos e estruturas exploráveis é de extrema importância para a criação de conteúdo de jogo que seja realmente interessante para o jogador.

Uma forma de atingir essa compreensão é através do estudo de casos de sucesso. Neste trabalho, o jogo Terraria é usado como estudo de caso central. Devido à escassez de documentação técnica pública detalhando o processo específico de construção de seus mapas, a pesquisa se justifica como um esforço de engenharia reversa experimental. O objetivo é reproduzir e compreender as etapas lógicas e a orquestração de algoritmos necessárias para gerar mundos com características topográficas e estruturais similares às observadas na referência.

## 1.2 OBJETIVOS

O **objetivo principal** deste trabalho é compreender as técnicas e algoritmos necessários para a geração procedural de mapas bidimensionais baseados em grade (*tile-based*) compostos por diferentes biomas, por meio da reprodução de características topográficas e estruturais do jogo Terraria.

Para alcançar o objetivo geral, foram definidos os seguintes objetivos específicos:

- implementar um sistema que integre múltiplos algoritmos de GPC para a criação de biomas e topografias bidimensionais similares ao jogo de referência;
- analisar o impacto da parametrização e da combinação de técnicas distintas na diversidade e na coerência estrutural dos cenários gerados;
- realizar uma comparação visual e estrutural entre os resultados obtidos e os mapas do jogo Terraria, validando a capacidade do sistema em reproduzir as características do estudo de caso.

O estudo presente neste trabalho limita-se estritamente à geração de mapas bidimensionais (2D) com vista lateral (*side-scrolling*), utilizando uma grade de *tiles* retangulares. Técnicas para terrenos 3D ou isométricos não são abordadas. Além disso, o trabalho concentra-se na geração estática do terreno e das estruturas (cavernas, biomas, calabouços). A geração de entidades dinâmicas, como NPCs (*Non-Playable Characters*), itens ou mecânicas de jogabilidade, foge ao escopo desta pesquisa.

## 1.3 ESTRUTURA DO TRABALHO

Este trabalho está organizado da seguinte forma:

- **Capítulo 2 (Fundamentos)**: apresenta a base teórica dos algoritmos utilizados;
- **Capítulo 3 (Trabalhos Relacionados)**: revisa a literatura acadêmica e técnica existente sobre geração procedural em jogos, comparando diferentes abordagens e metodologias;
- **Capítulo 4 (Desenvolvimento)**: descreve a estruturação do sistema criado, detalhando a sua implementação, as regras procedurais e algoritmos aplicados para a definição de cada bioma e as soluções encontradas para problemas de sobreposição e transição de terrenos;
- **Capítulo 5 (Resultados)**: demonstra os resultados visuais obtidos, comparando as saídas dos algoritmos com a referência e discutindo a eficácia das técnicas combinadas;

- **Capítulo 6 (Conclusão)**: sintetiza o que foi apresentado, discute as limitações encontradas e sugere caminhos para trabalhos futuros.

# 2 FUNDAMENTOS

Neste capítulo, são explorados quatro algoritmos essenciais, amplamente utilizados em várias áreas da Computação, e que foram aplicados na geração dos mapas apresentados neste trabalho. Cada um desses algoritmos oferece contribuições específicas para a criação de mapas detalhados e diversos, com aplicações que abrangem desde a modelagem de terrenos naturais e o processamento de imagens até o desenvolvimento de ambientes artificiais em jogos e simulações.

## 2.1 AUTÔMATOS CELULARES

Os autômatos celulares são uma abordagem muito utilizada na geração procedural de mapas bidimensionais. Esses sistemas são inspirados em fenômenos naturais, como o crescimento de colônias de bactérias, e simulam comportamentos complexos a partir de interações locais. O conceito foi inicialmente proposto por John von Neumann na década de 1940 e popularizado em 1970 pelo famoso "Jogo da Vida" de John Conway, apresentado ao grande público na coluna de Gardner (1970). Esse trabalho demonstrou que, mesmo com regras simples, é possível obter uma diversidade surpreendente de padrões e estruturas emergentes, característica ideal para a criação procedural de ambientes orgânicos como cavernas e terrenos.

Um autômato celular é um sistema composto por uma grade de células, que pode variar em dimensões (unidimensional, bidimensional ou multidimensional). Cada célula pode assumir um estado (como "cheio" ou "vazio"), e seu próximo estado é determinado por regras que consideram os estados das células vizinhas (Wolfram, 2002). No contexto deste trabalho, foca-se na grade bidimensional. Os tipos de vizinhança mais comuns usadas em autômatos celulares 2D são a **vizinhança de von Neumann**, que considera as células vizinhas nas direções ortogonais (norte, sul, leste e oeste), e a **vizinhança de Moore**, que amplia esse escopo ao incluir também vizinhos nas direções diagonais.

As regras de transição de estados definem como uma célula evolui em função das vizinhas. Um exemplo de regra simples para geração de terrenos ou cavernas pode ser encontrada no trabalho de Johnson et al (2010): se uma célula está preenchida e tem menos de 4 vizinhos preenchidos, ela se torna vazia; se está vazia e tem mais de 5 vizinhos preenchidos, ela se torna preenchida. Essa abordagem simula o preenchimento de grandes áreas enquanto permite a formação de pequenos espaços vazios. O processo de evolução das células é repetido várias vezes, até que a estrutura do mapa tenha atingido um estado desejável.

O Algoritmo 1 apresenta um pseudocódigo simples para ilustrar o funcionamento de um autômato celular aplicado à geração procedural de mapas. No início do processo, a grade é populada com valores aleatórios, onde cada célula tem uma probabilidade definida

de estar “cheia” ou “vazia”. Este estado inicial pode ser observado na Figura 1. A partir daí, a aplicação das regras de transição começa a moldar a distribuição de células, formando áreas mais definidas. Um exemplo de estado final pode ser observado na Figura 2.

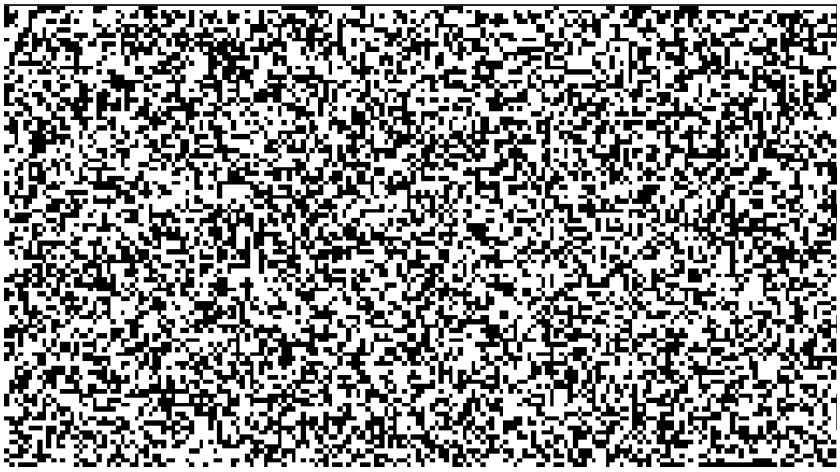
**Algoritmo 1:** Pseudocódigo de um autômato celular.

**Input:** Grade bidimensional $G$, número de iterações $n$, probabilidade de célula viva $p$

**Output:** Grade $G$ modificada com o mapa gerado

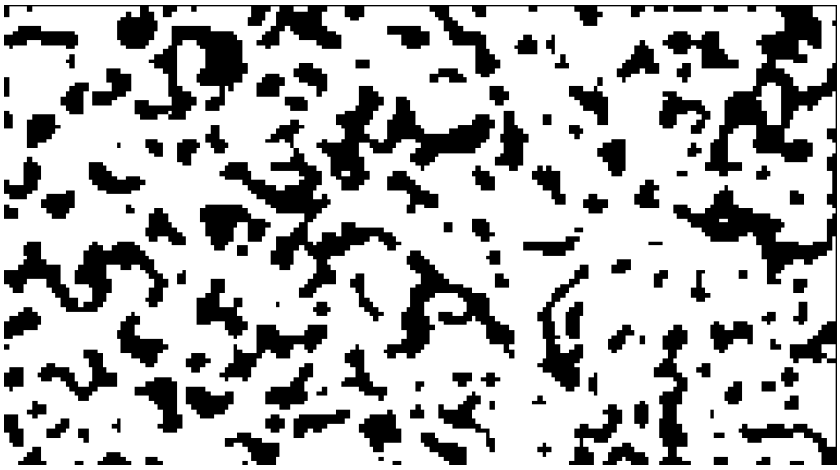
```
1  Passo 1: Inicialize a grade G com dimensões definidas
2  para cada célula C em G faça
3      se Aleatório() < p então
4          Marque C como viva;
5      senão
6          Marque C como morta;
7      fim se
8  fim para cada
9  Passo 2: Para n iterações, aplique as regras do autômato celular
10 para i ← 1 até n faça
11     para cada célula C em G faça
12         Conte o número de vizinhos vivos V ao redor de C;
13         se C está viva então
14             se V < 4 então
15                 Marque C como morta;
16             senão
17                 Mantenha C viva;
18             fim se
19         senão
20             se V > 5 então
21                 Marque C como viva;
22             fim se
23         fim se
24     fim para cada
25     Atualize a grade com as novas marcações;
26 fim para
27 Retorne G;
```

Figura 1 – Grade gerada aleatoriamente.



Fonte: Elaborado pelo autor. (2025).

Figura 2 – Grade após a aplicação do algoritmo.



Fonte: Elaborado pelo autor. (2025).

Em jogos, é comum o uso de autômatos celulares na geração procedural de mapas que envolvem a criação de ambientes subterrâneos ou irregulares. Na literatura de desenvolvimento de jogos, essa técnica é frequentemente descrita como o método padrão para a criação de níveis em jogos do gênero *roguelike* e para a simulação de erosão em cavernas, devido à sua capacidade de eliminar células isoladas e criar grandes aglomerados abertos (Johnson et al, 2010). Ao ajustar os parâmetros das regras, é possível criar mundos que variam de densamente povoados por cavernas a vastos espaços abertos (Shaker et al, 2016).

Embora os autômatos celulares sejam uma ferramenta eficaz para geração procedural, um dos principais desafios é encontrar o conjunto adequado de parâmetros e regras para garantir que o resultado final seja interessante e jogável. Mapas gerados por

autômatos celulares podem acabar sendo muito fragmentados ou muito densos, dependendo das condições iniciais e das regras escolhidas. Ajustes finos são necessários para garantir que o mapa gerado possua um bom equilíbrio entre áreas abertas e obstruções (Togelius et al, 2011).

## 2.2 RUÍDO DE PERLIN

O ruído de Perlin, criado por Perlin (1985), é frequentemente usado na geração procedural de texturas, terrenos e mapas bidimensionais. Um ruído pode ser descrito como uma função que gera variações pseudoaleatórias, sendo utilizado em computação gráfica para criar texturas ou padrões aparentemente aleatórios. O ruído de Perlin se destaca por ser um ruído de gradiente, o que significa que ele utiliza gradientes aleatórios pré-definidos em pontos específicos de uma grade e interpola suavemente os valores entre esses pontos. Essa interpolação cria transições contínuas e suaves, em vez de saltos abruptos, resultando em uma aparência mais natural e orgânica. Essa característica torna o ruído de Perlin especialmente adequado para simulações de paisagens, superfícies irregulares e outros fenômenos naturais (Lagae et al, 2010).

O funcionamento matemático do Ruído de Perlin baseia-se na definição de uma grade de vetores unitários de gradiente, orientados pseudoaleatoriamente nos vértices de células inteiras. Para determinar o valor de ruído em um ponto de coordenadas reais $(x, y)$, o algoritmo realiza três operações sequenciais fundamentais, conforme detalhado por Perlin (1985) e Ebert et al (2002). Inicialmente, calculam-se os vetores de distância entre o ponto $(x, y)$ e os quatro vértices da célula que o contém. Em seguida, realiza-se o **produto escalar** entre o vetor de gradiente de cada vértice e o respectivo vetor de distância, gerando quatro valores escalares de influência.

A etapa final é a interpolação desses valores. Diferente de uma interpolação linear simples, o ruído de Perlin aplica uma curva de suavização às coordenadas paramétricas locais $(u, v)$ antes da mistura. A função polinomial originalmente proposta é a função conhecida como smoothstep, descrita pela Equação 2.1:

$$f(t) = 3t^2 - 2t^3. \tag{2.1}$$

Embora eficaz, esta função não é ideal, pois sua derivada segunda é descontínua nas extremidades ($t = 0$ e $t = 1$), o que pode gerar artefatos visuais. Para corrigir isso, Perlin (2002) propôs uma função de suavização aprimorada, baseada em um polinômio de quinto grau, que garante continuidade de ordem superior. A equação atualizada, considerada o padrão moderno para implementação do ruído, é apresentada na Equação 2.2:

$$f(t) = 6t^5 - 15t^4 + 10t^3. \tag{2.2}$$

O Algoritmo 2 descreve a implementação básica do algoritmo do ruído de Perlin em 2D.

**Algoritmo 2:** Algoritmo do Ruído de Perlin

**Entrada:** Coordenadas $(x, y)$ no espaço

**Saída:** Valor de ruído em $(x, y)$

**Função** Gradiente(Aleatório, $x$, $y$)

    Retorne o produto escalar do vetor gradiente pseudo-aleatório com $(x, y)$;

**Função** InterpolaçãoSuave($t$)

    Retorne $t^3 \cdot (t \cdot (t \cdot 6 - 15) + 10)$;

**Passo 1**: Determine o quadrante no qual o ponto $(x, y)$ se encontra:

$x_0 \leftarrow \lfloor x \rfloor$, $y_0 \leftarrow \lfloor y \rfloor$;

$x_1 \leftarrow x_0 + 1$, $y_1 \leftarrow y_0 + 1$;

**Passo 2**: Calcule as distâncias entre o ponto $(x, y)$ e os pontos de grade:

$\Delta x \leftarrow x - x_0$, $\Delta y \leftarrow y - y_0$;

**Passo 3**: Gere vetores gradientes pseudo-aleatórios nos vértices do quadrante:

$g_{00} \leftarrow$ Gradiente($Aleatório(x_0, y_0), \Delta x, \Delta y$);

$g_{01} \leftarrow$ Gradiente($Aleatório(x_0, y_1), \Delta x, \Delta y - 1$);

$g_{10} \leftarrow$ Gradiente($Aleatório(x_1, y_0), \Delta x - 1, \Delta y$);

$g_{11} \leftarrow$ Gradiente($Aleatório(x_1, y_1), \Delta x - 1, \Delta y - 1$);

**Passo 4**: Interpole suavemente os valores calculados:

$sx \leftarrow$ InterpolaçãoSuave($\Delta x$);

$sy \leftarrow$ InterpolaçãoSuave($\Delta y$);

**Passo 5**: Realize a interpolação linear:

$n_0 \leftarrow (1 - sx) \cdot g_{00} + sx \cdot g_{10}$;

$n_1 \leftarrow (1 - sx) \cdot g_{01} + sx \cdot g_{11}$;

**Passo 6**: Interpole os dois valores anteriores:

ValorRuído $\leftarrow (1 - sy) \cdot n_0 + sy \cdot n_1$;

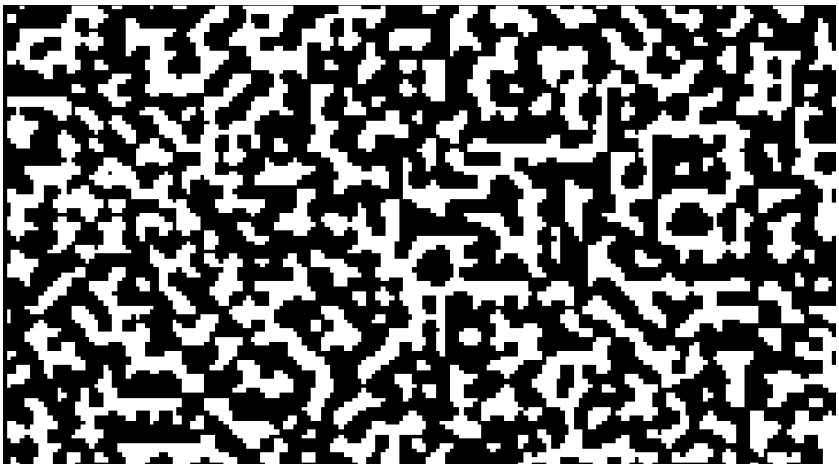
**Retorne** ValorRuído;

Uma das aplicações mais comuns do Ruído de Perlin é na geração procedural de terrenos. Em jogos eletrônicos e simulações, ele é utilizado para criar paisagens complexas com montanhas, vales e colinas, todos com transições suaves entre diferentes níveis de altura. Além disso, o Ruído de Perlin é amplamente empregado na geração de texturas para objetos tridimensionais, como madeira, nuvens e mármore (Ebert et al, 2002). Outra aplicação importante é em simulações físicas, onde o Ruído de Perlin pode ser utilizado para modelar fenômenos naturais como a movimentação de fluidos, a formação de nuvens, ou mesmo a geração de superfícies rugosas, como terrenos vulcânicos ou desertos.

A Figura 3 apresenta uma visualização gráfica de uma aplicação prática do algoritmo. Embora a função matemática base do ruído de Perlin gere valores contínuos e transições

mais suaves (originalmente gradientes em escala de cinza), para a definição de mapas baseados em grade, é necessário aplicar uma etapa de limiarização (*thresholding*) sobre o resultado bruto. Na imagem, os valores foram discretizados em apenas dois estados binários: preto (representando valores abaixo de um determinado limiar) e branco (valores acima). Esse processo de binarização transforma os gradientes suaves do algoritmo em fronteiras rígidas e abruptas, delimitando claramente áreas sólidas e vazias.

Figura 3 – Exemplo de grade gerada com o ruído de Perlin.



Fonte: Elaborado pelo autor. (2025).

Em grandes escalas, porém, o padrão do ruído pode se tornar repetitivo e previsível. A solução comum é combinar múltiplas camadas de ruído em diferentes frequências e amplitudes. A soma desses ruídos, gerando um ruído fractal, gera padrões mais variados e complexos (Fournier et al, 1982) que se assemelham a estruturas naturais.

## 2.3 ALGORITMO DE CAMINHADA ALEATÓRIA

O algoritmo de caminhada aleatória pode ser descrito como uma sequência de passos dados por um agente, em que, a cada iteração, a direção do movimento é escolhida de forma aleatória. Esse conceito é particularmente útil na geração procedural de mapas, pois permite que labirintos, cavernas e outros ambientes complexos sejam criados de maneira dinâmica e variada a partir de regras simples (Spitzer, 2001). No contexto de mapas bidimensionais, o agente se move por uma grade, onde cada célula representa uma posição possível no mapa. A cada passo, o agente escolhe uma direção – norte, sul, leste ou oeste – e se move para a célula adjacente correspondente.

O comportamento básico do algoritmo de caminhada aleatória é definido por sua imprevisibilidade, o que resulta em padrões únicos a cada execução do algoritmo, mas impõe desafios significativos ao controle da morfologia final do mapa. A simples limitação do número de passos atua apenas sobre a extensão da área percorrida, sem

afetar a aleatoriedade da trajetória. Para obter estruturas mais coerentes e direcionadas (como corredores ou salas definidas), é necessário empregar variações do algoritmo que introduzem **heurísticas de movimento**, como a utilização de diferentes probabilidades para privilegiar direções específicas, ou a aplicação de regras de inércia, que incentivam o agente a manter sua direção atual por múltiplos passos antes de virar (Rose e Bakaoukas, 2016; Dubský, 2022).

Em sua versão mais básica, o agente é inicialmente posicionado em uma célula aleatória da grade. A partir desse ponto, a cada iteração, ele escolhe uma direção de movimento com probabilidade uniforme entre as quatro direções cardeais. O processo continua até que o número de iterações definido pelo programador seja alcançado. Ao final do processo, a grade conterá um caminho aleatório, que pode ser utilizado para gerar cavernas, corredores ou até mesmo mapas completos.

Na Figura 4, é apresentado um exemplo visual de como um agente se move. O agente começa no canto superior direito da grade e, após 300 passos, cria um caminho aleatório que pode ser interpretado como um caminho contínuo e tortuoso. O pseudocódigo presente no Algoritmo 3 descreve de forma simplificada o funcionamento do algoritmo.

Figura 4 – Grade após aplicação do algoritmo de caminhada aleatória.

Fonte: Elaborado pelo autor. (2025).

**Algoritmo 3:** Algoritmo de caminhada aleatória

**Entrada:** Grade bidimensional $G$, número de passos $n$, posição inicial $(x_0, y_0)$
**Saída:** Grade $G$ com caminho gerado

**Passo 1**: Inicialize a posição inicial do agente $(x, y) \leftarrow (x_0, y_0)$;
Marque a posição inicial $(x_0, y_0)$ como parte do caminho;
**Passo 2**: Para $n$ iterações, mova o agente aleatoriamente
**para** $i \leftarrow 1$ *até* $n$ **faça**
    Escolha uma direção aleatória $d$ entre {norte, sul, leste, oeste};
    **se** $d$ *é norte* **então**
        $y \leftarrow y - 1$;
    **senão**
        **se** $d$ *é sul* **então**
            $y \leftarrow y + 1$;
        **senão**
            **se** $d$ *é leste* **então**
                $x \leftarrow x + 1$;
            **senão**
                $x \leftarrow x - 1$ ;  `/* Caso seja oeste */`
            **fim se**
        **fim se**
    **fim se**
    **se** *a nova posição* $(x, y)$ *está dentro dos limites da grade* **então**
        Marque $(x, y)$ como parte do caminho;
    **senão**
        Retorne para a posição anterior;
    **fim se**
**fim para**
**Retorne** $G$;

Esse algoritmo pode ser facilmente adaptado para incluir restrições adicionais, como evitar que o agente retorne para posições já visitadas ou limitar o movimento a áreas específicas da grade. Isso permite maior controle sobre o formato do mapa gerado, mantendo ainda a imprevisibilidade característica do algoritmo base.

O algoritmo de caminhada aleatória também pode ser combinado com outras técnicas de geração procedural, como o uso de autômatos celulares ou algoritmos baseados em grafos, para criar mapas mais ricos e variados. Essa abordagem híbrida tem sido explorada com sucesso tanto na literatura acadêmica quanto no desenvolvimento de jogos, visando mitigar as limitações individuais de cada algoritmo e potencializar a complexidade dos ambientes gerados (Shaker et al, 2016; Macedo e Chaimowicz, 2017).

## 2.4 ALGORITMO DE DESLOCAMENTO DO PONTO MÉDIO

O algoritmo de deslocamento do ponto médio (*midpoint displacement*) é uma técnica recursiva muito utilizada na computação gráfica para a geração de fractais aleatórios, sendo particulamente eficaz na modelagem de fenômenos naturais como linhas costeiras, silhuetas de montanhas e relâmpagos. A técnica foi popularizada no contexto de geração de terrenos pelo trabalho seminal de Fournier et al (1982), que propôs métodos eficientes para gerar aproximações do Movimento Browniano Fractal (fBm).

Diferente de abordagens baseadas em grade fixa (como autômatos celulares), o algoritmo do ponto médio opera subdividindo geometricamente um segmento. O princípio básico consiste em tomar um segmento de reta definido por dois pontos, encontrar o seu ponto médio e deslocá-lo perpendicularmente por um valor aleatório. Esse processo divide o segmento original em dois novos segmentos menores. O algoritmo é então aplicado recursivamente a cada novo segmento, reduzindo-se a magnitude do deslocamento aleatório a cada iteração.

A característica fractal do resultado, ou seja, a autossimilaridade estatística em diferentes escalas, é controlada pela taxa de decaimento do deslocamento aleatório. A amplitude do deslocamento $\Delta$ na iteração $i$ é definida pela Equação 2.3:

$$\Delta_i = \Delta_0 \cdot 2^{iH}, \tag{2.3}$$

onde $\Delta_0$ é a amplitude inicial e $H$ é o expoente de Hurst (ou parâmetro de rugosidade), variando tipicamente entre 0 e 1. Um valor de $H$ próximo de 1 resulta em uma curva suave (o deslocamento diminui rapidamente), enquanto um valor próximo de 0 gera uma curva extremamente rugosa e caótica.

Essa técnica é classificada como um método de refinamento de polígonos. Embora seja matematicamente capaz de gerar curvas suaves dependendo dos parâmetros e da técnica de interpolação final, sua implementação clássica utilizando interpolação linear entre os vértices gera formas com “quinas” vivas e angulações abruptas a cada subdivisão. Essa propriedade torna o algoritmo ideal para simular estruturas fraturadas, rochas quebradas ou, no contexto deste trabalho, ruínas antigas onde se deseja evitar a suavidade contínua típica de erosões hídricas simuladas por ruídos de gradiente (Fournier et al, 1982).

O Algoritmo 4 descreve a lógica recursiva para aplicações em uma dimensão (1D).

**Algoritmo 4:** Algoritmo de deslocamento do ponto médio (1D)

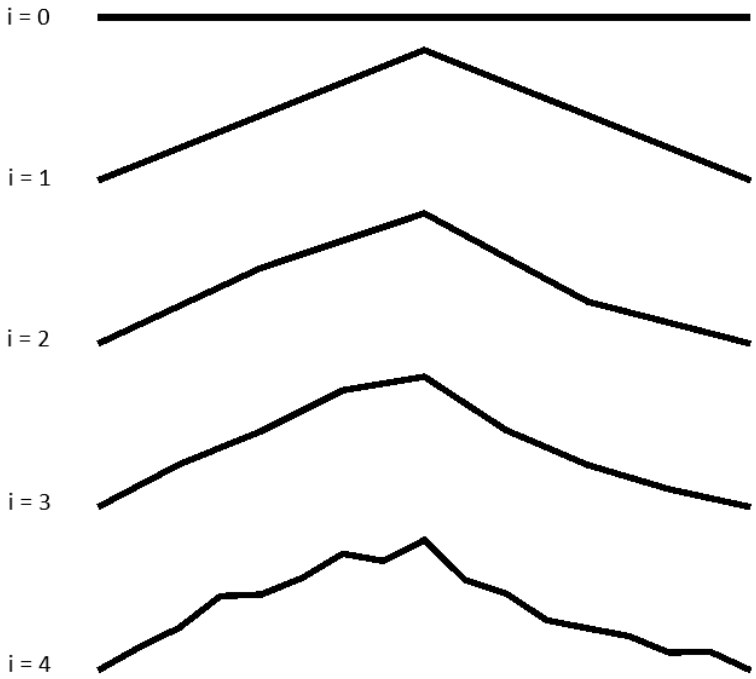
**Input:** Ponto inicial $P_A$, Ponto final $P_B$, Rugosidade $R$, Iterações $n$

**Output:** Lista de pontos que compõem a linha fractal

**Função** Subdividir($P_1, P_2, nivel$)
  **se** $nivel = 0$ **então**
    **retorna** lista contendo $\{P_1, P_2\}$;
  **fim se**
  `/* Calcula o ponto médio geométrico */`
  $P_{medio} \leftarrow (P_1 + P_2)/2$;
  `/* Calcula deslocamento aleatório proporcional ao nível atual */`
  $Magnitude \leftarrow R^{nivel}$;
  $Deslocamento \leftarrow Aleatorio(-Magnitude, Magnitude)$;
  $P_{medio}.y \leftarrow P_{medio}.y + Deslocamento$;
  `/* Chamada recursiva para os sub-segmentos */`
  $SegmentoEsq \leftarrow Subdividir(P_1, P_{medio}, nivel - 1)$;
  $SegmentoDir \leftarrow Subdividir(P_{medio}, P_2, nivel - 1)$;
  **retorna** Concatenar($SegmentoEsq, SegmentoDir$);
**Inicio**:
Resultado $\leftarrow$ Subdividir($P_A, P_B, n$);
**Retorne** Resultado;

A Figura 5 ilustra as primeiras iterações do algoritmo, demonstrando como a complexidade visual emerge da repetição de uma regra simples de subdivisão no ponto médio, com $\Delta_0 = 2$ e $H = 0.5$.

Figura 5 – Etapas iniciais do algoritmo de deslocamento do ponto médio.




Fonte: Elaborado pelo autor. (2025).

# 3 TRABALHOS RELACIONADOS

A Geração Procedural de Conteúdo é uma área de pesquisa consolidada, com décadas de evolução técnica voltada para a automação da criação de ambientes virtuais. A literatura acadêmica sobre o tema abrange desde métodos fundamentais de ruído e autômatos celulares até abordagens híbridas contemporâneas que buscam equilibrar aleatoriedade e controle de design. Nesta seção, revisam-se as principais contribuições da área, organizadas pela natureza dos algoritmos e seus objetivos de aplicação.

No campo da geração de terrenos e superfícies, o trabalho de Rose e Bakaoukas (2016) estabelece uma comparação crítica entre diferentes funções de ruído. Os autores concluem que, para mapas de jogos, algoritmos baseados em gradiente (como Perlin e Simplex) oferecem o melhor compromisso entre custo computacional e qualidade visual orgânica. No contexto de jogos de plataforma, Faruolo e Aguiar (2014) mostram como combinar o algoritmo de deslocamento do ponto médio para o relevo da superfície com a caminhada aleatória para escavar cavernas, criando uma morfologia que simula elementos naturais. Similarmente, Canedo (2022) valida o uso de ruídos procedurais estocásticos para garantir a variabilidade de ilhas e continentes em jogos 2D, destacando a escalabilidade dessas técnicas para grandes mapas.

Para a estruturação de ambientes subterrâneos e masmorras, onde a conectividade é prioritária, Leite e Lima (2015) propõem um método de geração recursiva para calabouços, focado em evitar a repetição de padrões através de uma ordem de criação aleatória e etapas de validação de coesão. Avançando na complexidade estrutural, Macedo e Chaimowicz (2017) introduzem uma abordagem híbrida que combina autômatos celulares multicamadas com curvas de Hilbert. O diferencial desse trabalho reside no uso das curvas de preenchimento de espaço para garantir a conectividade topológica entre as regiões geradas pelos autômatos, resolvendo o problema frequente de áreas isoladas inacessíveis ao jogador.

Salinas (2023) realizou uma análise quantitativa comparando a caminhada aleatória, os autômatos celulares e um outro algoritmo chamado Vermes de Perlin (*Perlin Worms*) para a geração de cavernas 2D. O estudo conclui que, embora os autômatos ofereçam o melhor equilíbrio visual para erosão, a caminhada aleatória é superior na garantia de conectividade de longos túneis. Em paralelo, Gellel e Sweetser (2020) propõem um sistema híbrido para jogos do gênero *roguelike* que utiliza gramáticas para a estrutura lógica e autômatos para o refinamento espacial. Focando na narrativa emergente, Sepanmaa (2024) investiga como esses algoritmos clássicos podem ser adaptados para jogos *roguelite*, onde a geração do mapa deve suportar a progressão da história.

Além da geometria, a simulação de ecossistemas e biomas representa uma fronteira importante. Dubský (2022) desenvolveu um sistema de simulação de mundo 2D que

integra geração de terreno com dinâmica de fluidos (líquidos e gases), utilizando o ruído Simplex, que é uma alternativa ao ruído de Perlin, para transições de biomas em tempo real. Aprofundando a questão climática, Piispa (2022) demonstra que a combinação de ruídos com simulação de erosão hidráulica produz terrenos visualmente superiores aos métodos estáticos. Especificamente sobre a distribuição de vegetação e climas, Fischer et al (2020) apresentam o sistema *AutoBiomes*, que utiliza mapas de temperatura e umidade sobrepostos para definir proceduralmente as zonas ecológicas, superando a monotonia de terrenos baseados em uma única camada de ruído.

Também no contexto de abordagens híbridas, Babin (2025) propõe a união de técnicas tradicionais de GPC com modelos avançados de Inteligência Artificial. O autor destaca que o futuro da área não reside apenas na criação de *assets*, mas na segurança e robustez desses sistemas, garantindo que a geração autônoma respeite restrições de integridade e jogabilidade sem supervisão humana constante.

O presente trabalho tem como objetivo utilizar uma abordagem híbrida para a construção de mapas similares aos do jogo Terraria. No entanto, ao buscar na literatura acadêmica referências específicas sobre a arquitetura de geração de mundos do jogo Terraria, que é amplamente reconhecido pela complexidade de seus biomas e camadas verticais, nota-se uma significativa escassez de publicações formais. Não foram encontrados trabalhos que dissequem, com rigor científico, o processo completo de geração deste jogo específico ou que documentem metodologias para reproduzir sua estrutura geológica detalhada.

Atualmente, o conhecimento técnico aprofundado sobre esses processos reside majoritariamente em documentações informais da comunidade (*wikis*), fóruns de desenvolvimento e análises empíricas. Um exemplo primordial é a documentação do projeto tModLoader (tModLoader Team, 2016), que descreve os passos da geração de um mundo original (*vanilla*) do Terraria. O documento lista 107 etapas consecutivas, que vão desde a definição inicial da topologia do terreno, passando pela inserção de minérios e criação de biomas, até os acabamentos estéticos finais.

Similarmente, discussões em comunidades de desenvolvedores, como os fóruns da Unity, demonstram o esforço prático da indústria em replicar esse estilo de geração. Desenvolvedores debatem empiricamente a combinação de ruídos como Perlin, Simplex e ruído fractal para atingir resultados semelhantes aos do jogo (Unity Community, 2014), muitas vezes sem o respaldo de uma análise teórica formalizada.

Desta forma, o presente trabalho busca preencher essa lacuna na literatura. O objetivo não é apenas aplicar algoritmos conhecidos para tentar reproduzir a geração original do jogo, mas também sistematizar academicamente o conhecimento disperso em fóruns e documentações técnicas. Propõe-se, portanto, investigar e implementar uma arquitetura de geração procedural inspirada nas mecânicas observadas no Terraria, validando formalmente como a combinação de ruído de Perlin, autômatos celulares e

caminhada aleatória pode ser orquestrada para reproduzir a complexidade de camadas e biomas que caracterizam esse estilo de jogo.

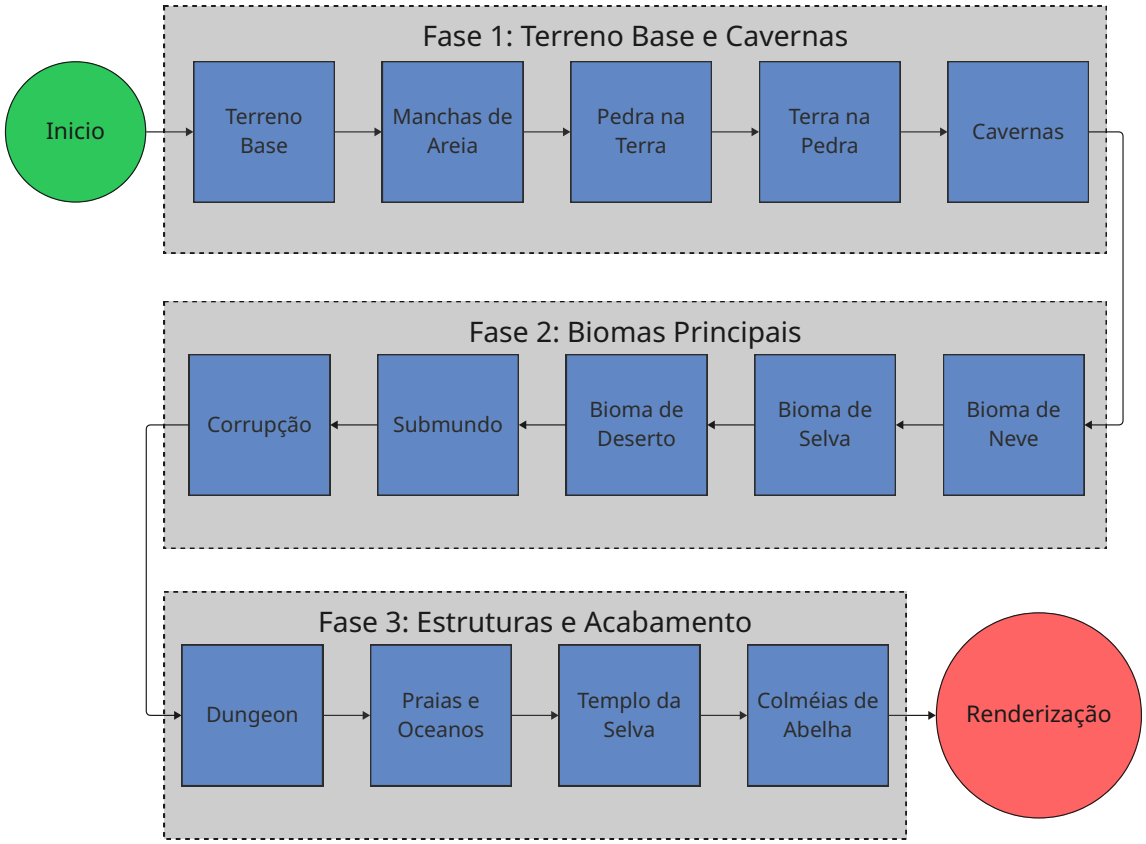
# 4 DESENVOLVIMENTO

Este capítulo apresenta os principais desafios encontrados durante o desenvolvimento do sistema de geração procedural de mapas bidimensionais e as soluções técnicas adotadas para resolvê-los. A implementação foi dividida em etapas progressivas, cada uma construindo sobre a anterior, permitindo a criação de ambientes complexos e variados.

## 4.1 VISÃO GERAL DO SISTEMA

O sistema foi desenvolvido no motor de jogos Unity utilizando a linguagem C#. O mapa é representado por uma matriz bidimensional de enumerações (`TileType[,] mapData`), em que cada elemento representa um *tile*. A geração segue um fluxo estrito de dependências, onde cada etapa constrói ou subtrai elementos da etapa anterior. Conforme ilustrado na Figura 6, o processo inicia com a definição da topologia do terreno e avança progressivamente para a escavação de cavernas, modificação de biomas e, finalmente, a inserção de estruturas artificiais selecionadas. Essa ordem assegura, por exemplo, que estruturas não sejam geradas no ar e que cavernas respeitem os limites dos biomas.

Figura 6 – Fluxograma da ordem das etapas da geração.




Fonte: Elaborado pelo autor. (2025).

Como os algoritmos implementados — especificamente o autômato celular e o ruído de Perlin — operam sobre a totalidade da grade e exigem verificação constante de vizinhança, a matriz bidimensional foi a escolha natural para o armazenamento dos *tiles* do mapa.

Devido às dimensões dos mapas (até 4200x1200 tiles), a geração completa demanda tempo considerável. Assim, foi implementado um sistema de execução por etapas que pode ser ativado quando necessário. Quando ativo, o gerador pausa após cada algoritmo, permitindo a inspeção visual isolada de cada etapa (terreno, cavernas, minérios) e a identificação de falhas lógicas que seriam invisíveis no resultado final mesclado.

Por fim, para garantir a reprodutibilidade dos mapas, o sistema permite especificar uma semente para o gerador de números pseudoaleatórios na inicialização, garantindo que a mesma sequência de decisões algorítmicas seja tomada, resultando em um mapa idêntico para a mesma entrada.

## 4.2 GERAÇÃO DO TERRENO BASE

Para a criação do relevo natural, o principal desafio foi gerar um terreno que apresentasse variações em múltiplas escalas, desde grandes colinas até pequenas irregularidades. Para isso, utilizou-se uma composição de múltiplas frequências do ruído de Perlin, aplicando o princípio de ruído fractal (Perlin, 1985).

Três camadas de ruído foram utilizadas para compor o mapa de alturas do terreno, sendo que cada camada possui frequência e amplitude distintas. As colinas representam as grandes formações, obtidas por uma camada com baixa frequência e alta amplitude. Outra camada, com frequência e amplitude médias, adicionam um nível de variação adicional à camada anterior. Por fim, uma terceira camada, com alta frequência e baixa amplitude, é incluída para criar micro-variações. Esta abordagem permite controle independente de cada escala de detalhe por meio de parâmetros ajustáveis, facilitando a obtenção de diferentes estilos de terreno (Accidental Noise Library, 2011).

Além da altura, é feito um tratamento para a transição entre materiais (como terra e pedra), pois cortes abruptos resultam em uma aparência artificial. O Algoritmo 5 descreve o cálculo da profundidade dessa transição para uma determinada coluna $x$, utilizando um ruído unidimensional para criar a ondulação.

Dessa forma, pode-se garantir que a transição entre materiais seja suave e natural, seguindo padrões orgânicos similares aos encontrados em formações geológicas reais, e em consonância com as caracteristicas do mapa gerado pelo Terraria, como será visto mais adiante nos resultados. A Figura 7 mostra um exemplo do mapa que é gerado nessa etapa.

**Algoritmo 5:** Cálculo da profundidade de transição entre materiais.

**Input:** Coordenada horizontal $x$, $profundidade\_base$, $escala$, $variacao$
**Output:** Profundidade $y$ onde ocorre a transição

**Passo 1**: Calcule o valor do ruído para a coluna atual
$ruido \leftarrow Perlin(x/escala)$;
**Passo 2**: Aplique a variação à profundidade base
$deslocamento \leftarrow ruido \times variacao$;
$y \leftarrow profundidade\_base + deslocamento$;
**Retorne** $y$;

Figura 7 – Exemplo de saída da etapa de geração do terreno base.

Fonte: Elaborado pelo autor. (2025).

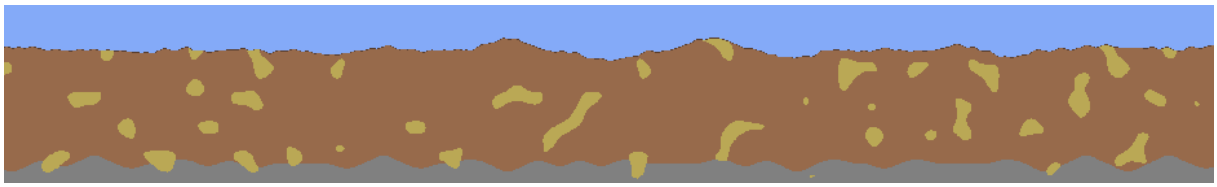
## 4.3 SISTEMA DE MATERIAIS SECUNDÁRIOS

A distribuição de materiais secundários, como areia ou depósitos minerais, não poderia ser uniforme. Implementou-se um sistema de amostragem com um limiar, em que a presença do material é determinada se o valor do ruído de Perlin em $(x, y)$ superar um valor predefinido. Assim, consegue-se limitar a quantidade e a densidade de manchas que aparecem, mas mantendo o formato que provém do ruído de Perlin.

Para evitar que manchas de superfície (como areia) se estendam irrealisticamente para as profundezas, introduziu-se um multiplicador de decaimento (*falloff*) baseado na profundidade. Esse multiplicador é aplicado ao valor do ruído antes da comparação com o limiar, criando uma transição suave entre a zona de alta densidade na superfície e a ausência total do material em camadas profundas. O processo é detalhado no Algoritmo 6. A Figura 8, que ilustra um recorte do mapa, com a primeira coordenada referente ao canto inferior esquerdo e a segunda ao canto superior direito, mostra o mapa obtido após essa etapa, com destaque nas manchas geradas. Todas as demais figuras neste e no próximo capítulo que representam recortes do mapa têm as suas coordenadas especificadas com o

mesmo formato de intervalo.

Figura 8 – Exemplo de geração dos depósitos de areia. Região do mapa ilustrada: $[(1072, 725), (3124, 1026)]$



Fonte: Elaborado pelo autor. (2025).

Além disso, bolsões de pedra e de terra são inseridos nas regiões opostas, ou seja, de terra e pedra, respectivamente. Ambos são gerados com processos similares ao das manchas de areia, com exceção do decaimento. No caso dos bolsões de pedra, foi implementado um sistema de estratificação em duas zonas distintas, separadas dinamicamente por uma fronteira ondulada (`deepSurfaceBoundary`):

**Algoritmo 6:** Cálculo do multiplicador de decaimento (*Falloff*).

**Input:** Profundidade atual $y$, profundidade inicial $y_{start}$, altura do falloff $h$
**Output:** Multiplicador $m \in [0, 1]$

**Passo 1**: Normalize a profundidade relativa
$t \leftarrow (y - y_{start})/h$;
**Passo 2**: Restrinja o valor entre 0 e 1 (Clamp)
**se** $t < 0$ **então**
    $m \leftarrow 0$;
**senão**
    **se** $t > 1$ **então**
        $m \leftarrow 1$;
    **senão**
        $m \leftarrow t$;
    **fim se**
**fim se**
**Retorne** $m$;

- **Zona Rasa:** localizada logo abaixo da superfície, utiliza uma frequência de ruído mais alta (`shallowStonePatchNoiseScale`), resultando em pequenos agrupamentos de pedras esparsas, simulando detritos e rochas soltas;
- **Zona Profunda:** localizada próxima à transição para a camada de pedra maciça, utiliza uma frequência menor (`deepStonePatchNoiseScale`) e um limiar de

preenchimento mais agressivo, gerando grandes maciços rochosos.

Essa abordagem cria um gradiente visual, sinalizando organicamente ao jogador a transição entre o solo macio da superfície e o subsolo rochoso.

## 4.4 GERAÇÃO DE CAVERNAS

No contexto da escavação de cavernas, inicialmente tentou-se utilizar algoritmos de caminhada aleatória simples, enviesados para um movimento horizontal, mas isso resultava em túneis excessivamente retilíneos. Para obter a sinuosidade característica de cavernas naturais, foram implementados *walkers* guiados pelo ruído de Perlin. Como refinamento adicional, o raio de escavação varia a cada passo dentro de um intervalo $[r_{\min}, r_{\max}]$. Esta combinação de direção guiada por campo vetorial e raio variável resulta em cavernas com aparência orgânica e imprevisível. O Algoritmo 7 demonstra como a direção do escavador é determinada a cada passo baseada em sua posição atual. A Figura 9 mostra algumas cavernas obtidas através desse método.

**Algoritmo 7:** Cálculo da direção de caminhamento via ruído de Perlin.

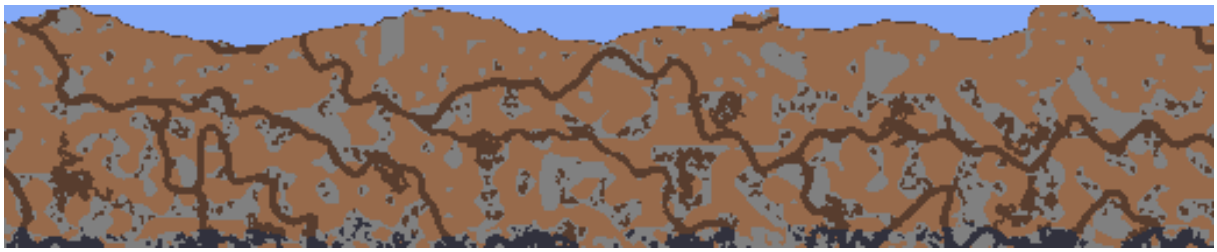
**Input:** Posição atual $(x, y)$, frequência do ruído $f$
**Output:** Vetor de direção $\vec{d}$

**Passo 1**: Obtenha o valor do ruído para a posição
$valor \leftarrow Perlin(x \times f, y \times f)$;
**Passo 2**: Mapeie o ruído para um ângulo em radianos
$\theta \leftarrow valor \times 2\pi$;
**Passo 3**: Converta o ângulo para um vetor unitário
$dx \leftarrow \cos(\theta)$;
$dy \leftarrow \sin(\theta)$;
$\vec{d} \leftarrow (dx, dy)$;
**Retorne** $\vec{d}$;

Para a geração de grandes câmaras interconectadas, presentes na camada mais profunda do mapa, aplicou-se um autômato celular padrão com as seguintes regras:

- **Semeadura inicial:** distribuição aleatória com probabilidade $p$ de cada célula estar "viva" (bloco);
- **Vizinhança de Moore:** considera os 8 vizinhos adjacentes, incluindo diagonais;
- **Regras de evolução:**
  - célula sólida com menos de 4 vizinhos sólidos → vira ar (erosão);
  - célula vazia com mais de 4 vizinhos sólidos → vira sólida (crescimento).

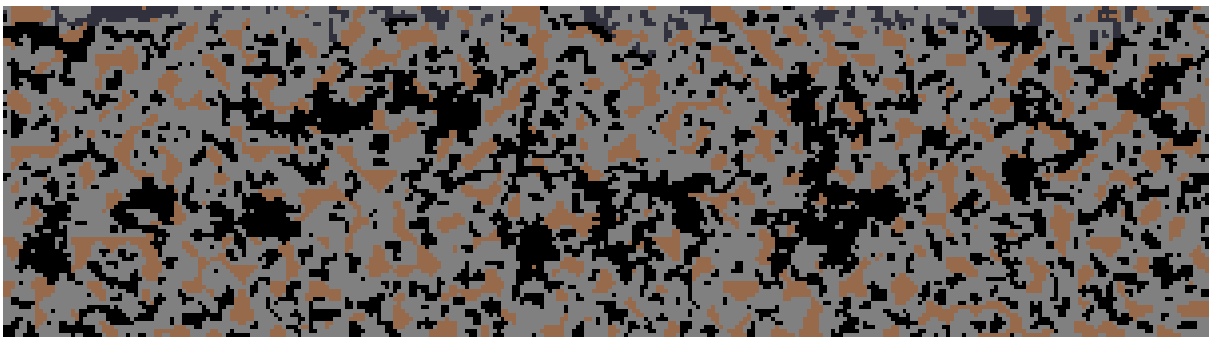
Figura 9 – Exemplo de cavernas geradas pelo algoritmo de caminhada aleatória adaptado. Região do mapa ilustrada: $[(1828, 738), (2980, 956)]$



Fonte: Elaborado pelo autor. (2025).

Para evitar a homogeneização excessiva das formas, limitou-se o algoritmo a 5 iterações. A ideia é balancear a criação de espaços abertos navegáveis com a manutenção de rugosidade nas paredes, se aproximando da estética de caverna natural (Fournier et al, 1982). A Figura 10 demonsta um resultado obtido ao se utilizar autômato celular nas cavernas mais profundas.

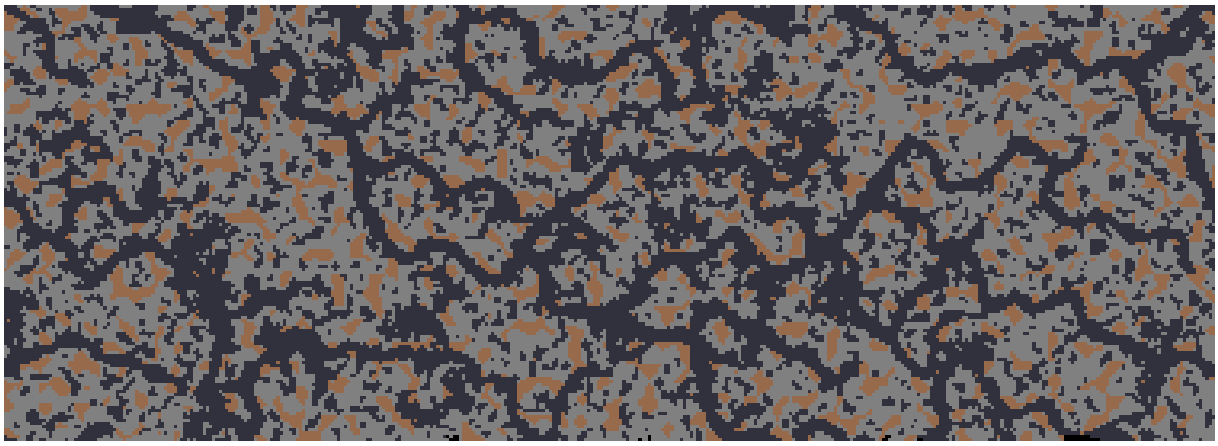
Figura 10 – Exemplo de geração de cavernas com o autômato celular. Região do mapa ilustrada: $[(0, 0), (661, 1026)]$



Fonte: Elaborado pelo autor (2025).

Para gerar uma interconexão maior entre tais camâras, utilizou-se também de *walkers* com movimento aleatório e com tempo de vida menor. Esses agentes, programados para escavar túneis por um número limitado de passos, atuam "rompendo" as paredes que separam os grandes salões isolados gerados previamente. Essa etapa de pós processamento é crucial para garantir a navegabilidade geral do subsolo profundo, assegurando que o jogador não encontre áreas vastas porém inacessíveis. O resultado visual desta abordagem híbrida pode ser observado na Figura 11.

Figura 11 – Exemplo de cavernas após o pós processamento com a Caminhada Aleatória. Região do mapa ilustrada: $[(3280, 182), (4110, 479)]$



Fonte: Elaborado pelo autor (2025).

## 4.5 GERAÇÃO DE BIOMAS

Os passos descritos nas seções anteriores são mais gerais e podem facilmente ser adaptados para contextos de outros jogos. Já esta seção descreve como foram gerados alguns biomas específicos do Terraria. Diferentes biomas exigiram abordagens geométricas distintas para simular suas condições climáticas e geológicas.

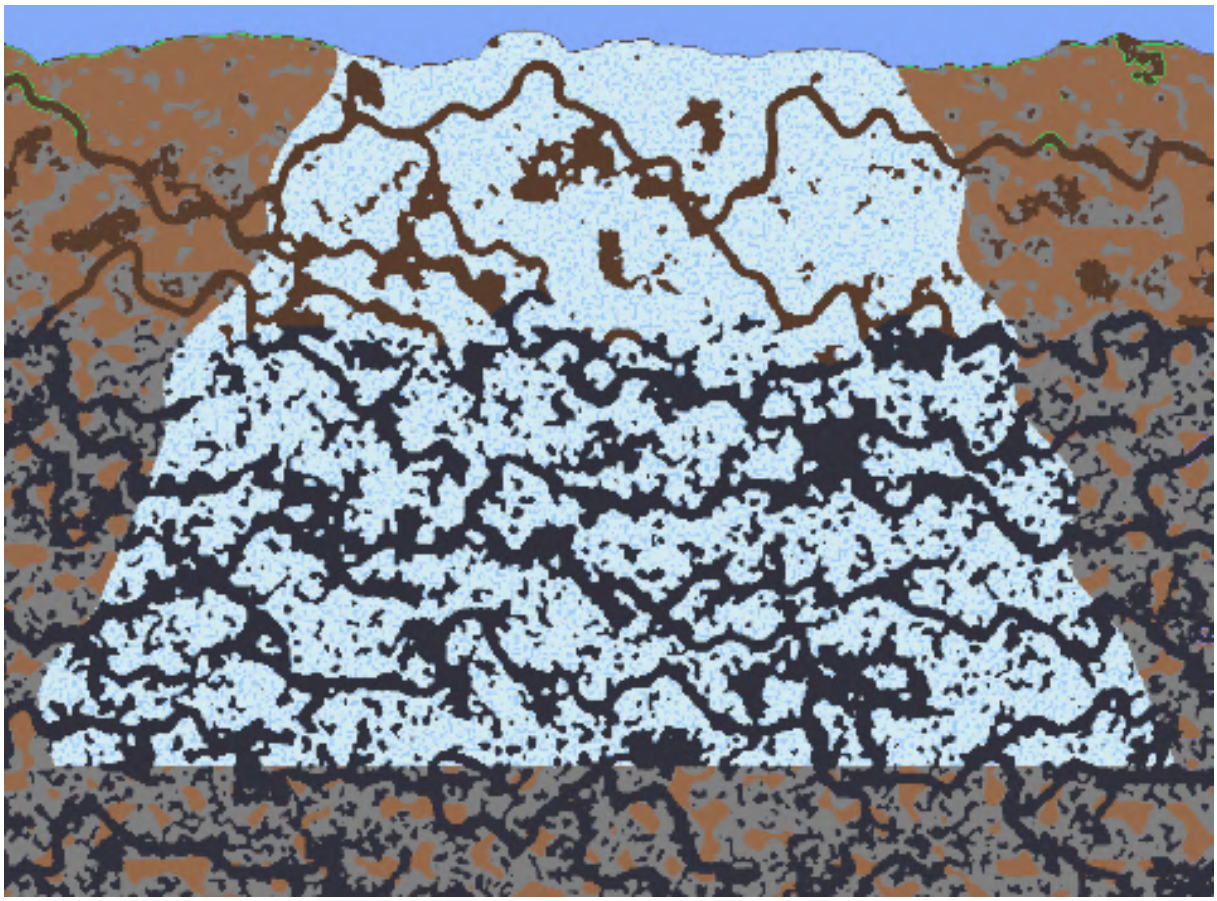
Para o **Bioma de Neve**, era necessário um formato trapezoidal (mais estreito na superfície, largo na profundidade), simulando a propagação de condições climáticas. Implementou-se uma interpolação baseado na altura, descrito no Algoritmo 8, que utiliza uma função quadrática para suavizar a expansão. O uso do termo quadrático $t^2$ cria uma curva de aceleração suave, tornando a expansão do bioma mais gradual próximo à superfície e acentuada no fundo. Além disso, foi aplicado o ruído de Perlin 1D em suas bordas laterais para remover o formato retilíneo não natural original. A Figura 12 mostra um exemplo de geração do bioma de neve com o formato trapezoidal e com bordas suavizadas.

Para o **Bioma da Selva**, foi utilizado o formato de um retângulo na vertical, com suas bordas sendo suavizadas também pelo Ruído de Perlin 1D. Porém, apesar de mais simples no seu formato, o bioma se destaca pelas estruturas que são inseridas nele posteriormente. A Figura 13 apresenta um exemplo da geração do bioma de selva.

Já para o **Bioma de Deserto**, buscou-se uma forma de superelipse com topo achatado. Utilizou-se um expoente variável na equação da elipse: fator 4.0 para a metade superior (criando o topo plano) e fator 2.0 para a base (arredondada). Isso permite controle preciso sobre a silhueta do bioma, facilitando a transição entre diferentes zonas climáticas.

Para simular túneis de arenito erodido no deserto, adotou-se o ruído de Perlin com

Figura 12 – Exemplo de geração do bioma trapezoidal da neve. Região do mapa ilustrada: $[(2632, 374), (3359, 982)]$



Fonte: Elaborado pelo autor (2025).

**Algoritmo 8:** Cálculo da largura do bioma de neve por profundidade.

**Input:** Profundidade atual $y$, $y_{superficie}$, $profundidade\_maxima$, $largura\_topo$, $largura\_base$

**Output:** Largura do bioma $L$ na profundidade $y$

**Passo 1**: Calcule o fator de interpolação normalizado

$distancia \leftarrow y_{superficie} - y$;

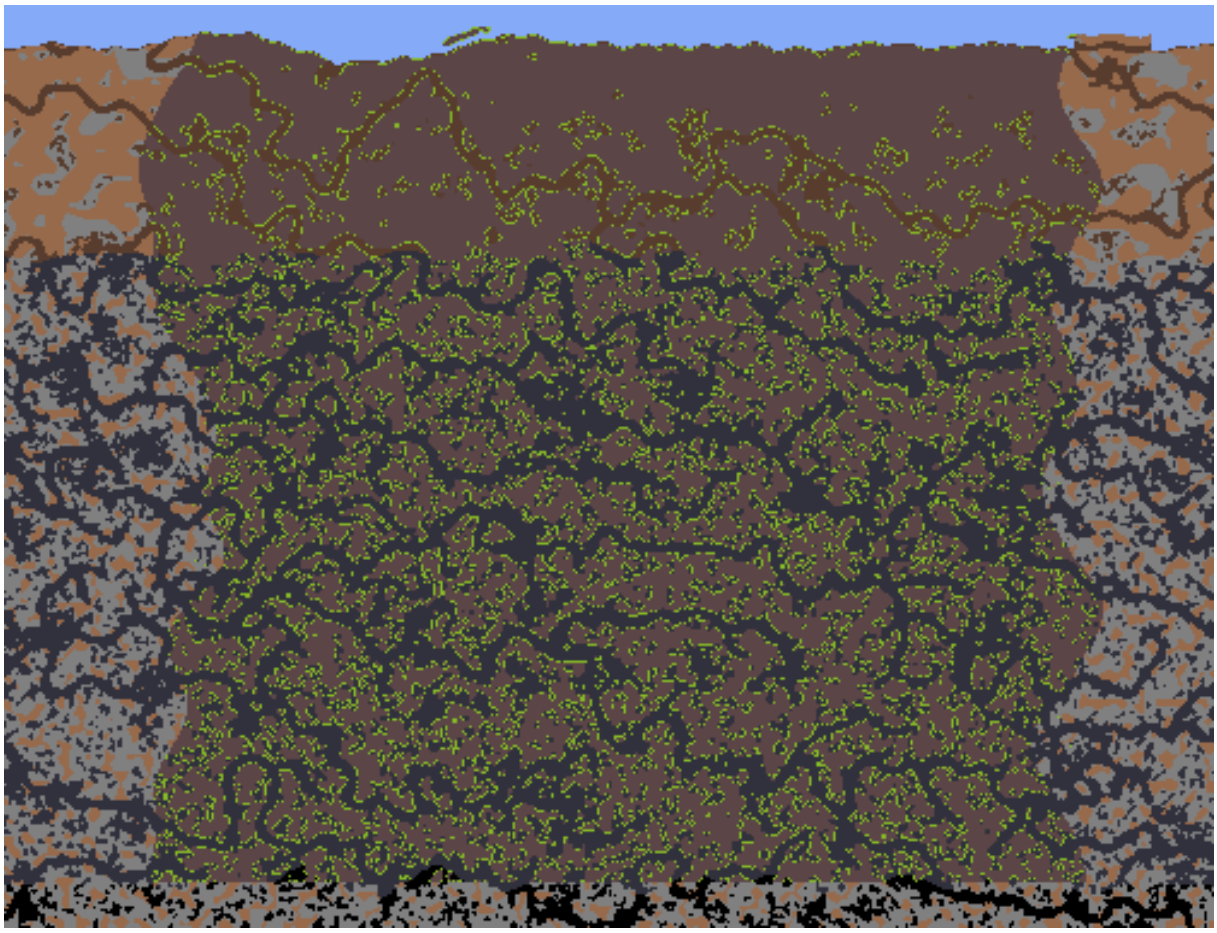
$t \leftarrow distancia / profundidade\_maxima$;

**Passo 2**: Interpole entre as larguras

$L \leftarrow largura\_topo + (largura\_base - largura\_topo) \times t^2$;

**Retorne** $L$;

Figura 13 – Exemplo de geração do bioma retangular da selva. Região do mapa ilustrada: $[(460, 136), (1559, 980)]$



Fonte: Elaborado pelo autor (2025).

transformação *ridged* (estriado). O processo de decisão para escavação de um bloco é descrito no Algoritmo 9. O operador `abs(valor - 0.5)` transforma o ruído suave em "cristas", onde os valores próximos de 0.5 (centro da distribuição) se tornam túneis, e valores extremos (próximos de 0 ou 1) se tornam paredes. Os parâmetros críticos desse método são: `threshold`, que controla a largura dos túneis (valores menores resultam em túneis mais largos); `frequência`, que controla a escala dos padrões (valores menores resultam em túneis mais longos e sinuosos); as `oitavas`, que adicionam detalhes em múltiplas escalas. A Figura 14 demonstra um resultado obtido na geração das cavernas do deserto.

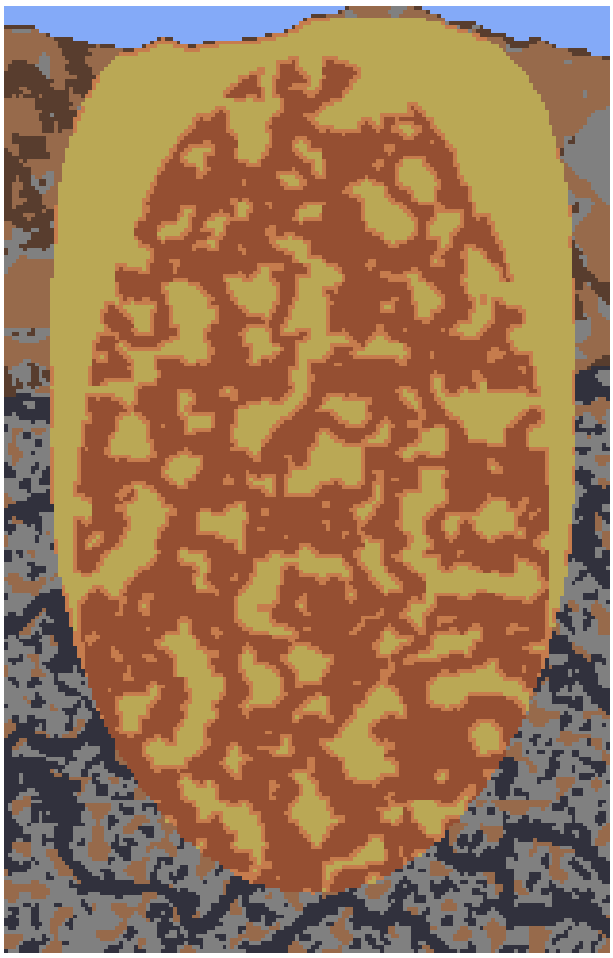
No caso do **Submundo**, a topografia é invertida para criar um teto plano de cinzas, limitando-se a altura das cavernas comuns. A complexidade do bioma reside nas formações verticais que representam o deslocamento de lava no cenário. Para esta etapa, foram combinadas duas técnicas distintas: para as quedas de lava, usou-se agentes de caminhada aleatória com viés vertical e desvios laterais estocásticos; e para os poços de lava, foram usados autômatos celulares com regras de evolução específicas. Complementarmente, veios

**Algoritmo 9:** Algoritmo de ruído *ridged* para túneis.

**Input:** Coordenadas $(x, y)$, frequência $f$, oitavas $o$, amplitude $A$, limiar $L$
**Output:** Tipo do bloco (Parede ou Túnel)

**Passo 1**: Amostre o ruído fractal
$valor \leftarrow MultiOctavePerlin(x, y, f, o, A)$;
**Passo 2**: Aplique a transformação *ridged* ("Crista")
$crista \leftarrow |valor - 0.5|$;
**Passo 3**: Verifique o limiar para escavação
**se** $crista > L$ **então**
    **retorna** Parede (Arenito);
**senão**
    **retorna** Túnel (Ar);
**fim se**

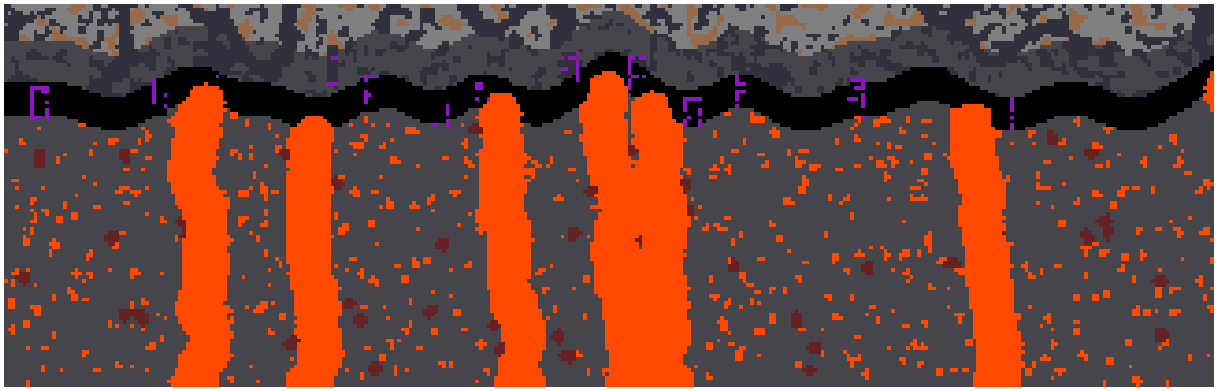
Figura 14 – Exemplo de geração do bioma do deserto. Região do mapa ilustrada: $[(1822, 358), (2170, 960)]$



Fonte: Elaborado pelo autor. (2025).

de minério específicos são inseridos por *walkers* com alta taxa de variação direcional, criando aglomerados irregulares nas cinzas. O resultado dessa composição pode ser visualizado na Figura 15. A lógica de geração das quedas de lava pode ser vista no Algoritmo 10, enquanto as características dos autômatos celulares usados na geração dos poços de lava estão descritas no Algoritmo 11.

Figura 15 – Exemplo de geração do bioma do submundo. Região do mapa ilustrada: $[(1796, 0), (2491, 237)]$



Fonte: Elaborado pelo autor. (2025).

**Algoritmo 10:** Geração de Quedas de Lava (Caminhada Aleatória Vertical).

**Input:** Limite superior $Y_{topo}$, largura $W$, número de fontes $N$

**Output:** Mapa com rios verticais de lava

```
1  enquanto rios gerados < N faça
2      x ← Aleatorio(0, W); se coluna x não está ocupada então
3          pos ← (x, Y_topo[x]); enquanto pos.y > 0 faça
4              Escavar(pos, Lava); direcao ← (0, −1);
               /* Aplica perturbação lateral aleatória                    */
5              se Aleatorio() < chance_jitter então
6                  direcao.x ← (Aleatorio() < 0.5)? − 1 : 1;
7              fim se
8              pos ← pos + direcao;
9          fim enqto
10         MarcarColunasOcupadas(x, raio);
11     fim se
12 fim enqto
```

**Algoritmo 11:** Geração de Poças de Lava (Autômato Celular).

**Input:** Mapa de Cinzas $G$, iterações $n$

**Output:** Mapa com poças de lava consolidadas

**Passo 1**: Semeadura Inicial **para cada** *bloco B em G do tipo Cinzas* **faça**
  **se** *Aleatorio()* $<$ *chance_semente* **então**
    $B \leftarrow Lava$;
  **fim se**
**fim para cada**

**Passo 2**: Suavização (Repetir $n$ vezes) **para cada** *bloco B em G* **faça**
  $vizinhos \leftarrow ContarVizinhosLava(B)$; **se** $B == Cinzas$ ***E*** $vizinhos > 1$ **então**
    $B \leftarrow Lava$ `// Regra de Crescimento`
  **senão**
    **se** $B == Lava$ ***E*** $vizinhos < 2$ **então**
      $B \leftarrow Cinzas$ `// Regra de Morte`
    **fim se**
  **fim se**
**fim para cada**

Por fim, o **Bioma de Corrupção** apresenta um desafio topológico distinto: a criação de abismos verticais profundos que conectam a superfície ao subsolo. Diferente das cavernas sinuosas geradas por *walkers* tradicionais, estes abismos exigem uma diretriz vertical forte, porém com variações laterais para manter o aspecto orgânico. Para alcançar este formato, implementou-se um escavador vertical cuja posição horizontal ($x$) é modulada por uma função senoidal em relação à profundidade ($y$). Isso cria paredes onduladas consistentes. O Algoritmo 12 detalha a lógica utilizada para definir a trajetória desses abismos.

Além dos abismos verticais, o algoritmo conecta o fundo dessas estruturas através de um túnel horizontal, garantindo que o jogador possa transitar entre os abismos sem retornar à superfície. A composição do bioma é finalizada substituindo a terra e a pedra da região afetada por um *tile* que representa pedra corrompida. A Figura 16 demonstra a estrutura vertical gerada.

**Algoritmo 12:** Geração de abismos verticais ondulados (Corrupção).

**Input:** Posição inicial $(x_0, y_0)$, profundidade máxima $D$, frequência $f$, amplitude $A$

**Output:** Túnel vertical escavado na grade $G$

**Passo 1**: Inicialize o escavador $y_{atual} \leftarrow y_0$;

**Passo 2**: Escave enquanto não atingir a profundidade alvo

**enquanto** $y_{atual} < (y_0 + D)$ **faça**

`/* Calcula o deslocamento lateral via função seno */`

$deslocamento \leftarrow \sin(y_{atual} \times f) \times A$;$x_{centro} \leftarrow x_0 + deslocamento$;

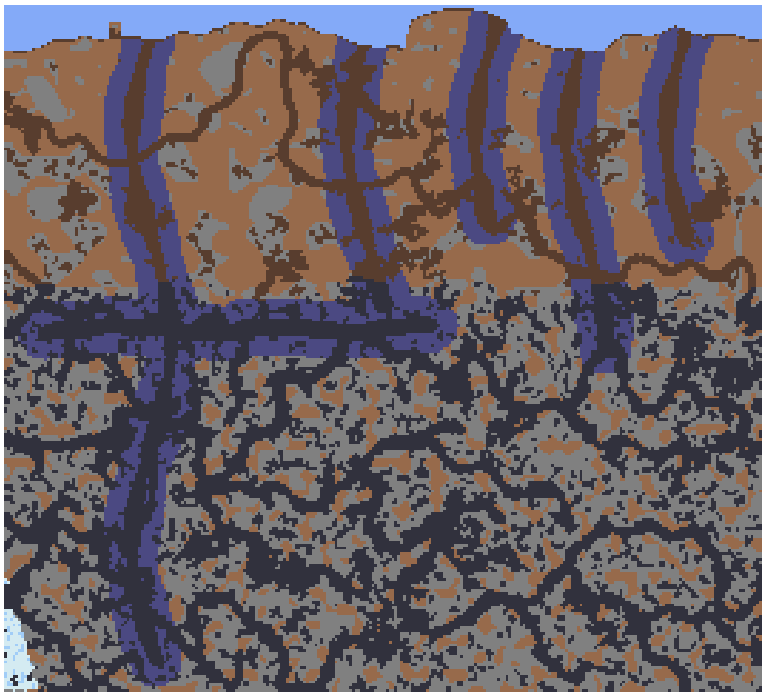
`/* Escava uma área circular ao redor do centro calculado */`

EscavarCirculo($G$, $x_{centro}$, $y_{atual}$, $Raio$);

$y_{atual} \leftarrow y_{atual} + 1$;

**fim enqto**

Figura 16 – Exemplo de geração do bioma de Corrupção. Região do mapa ilustrada: $[(3330, 435), (3915, 958)]$



Fonte: Elaborado pelo autor. (2025).

## 4.6 SISTEMA DE TÚNEIS COM COMPORTAMENTO COMPLEXO

Para a área da masmorra (*dungeon*), utilizou-se um caminhamento baseado em vetores de ponto flutuante aleatórios, permitindo movimento em qualquer ângulo, não

apenas nas direções cardinais, conforme o Algoritmo 13.

**Algoritmo 13:** Geração de direção aleatória livre.

**Input:** Ângulo mínimo $\alpha_{min}$, ângulo máximo $\alpha_{max}$
**Output:** Vetor de direção $\vec{v}$

**Passo 1**: Sorteie um ângulo no intervalo desejado
$\alpha \leftarrow Aleatorio(\alpha_{min}, \alpha_{max})$;
**Passo 2**: Converta graus para radianos
$\theta \leftarrow \alpha \times \frac{\pi}{180}$;
**Passo 3**: Calcule o vetor unitário
$\vec{v} \leftarrow (\cos(\theta), \sin(\theta))$;
**Retorne** $\vec{v}$;

Para evitar monotonia, os escavadores operam em uma máquina de estados com três comportamentos:

- **Modo Normal:** movimento com pequena oscilação e ocasionais mudanças de direção;
- **Modo Zig-Zag:** alternância sistemática entre dois ângulos diagonais;
- **Modo Labirinto:** movimento predominantemente lateral com ramificações.

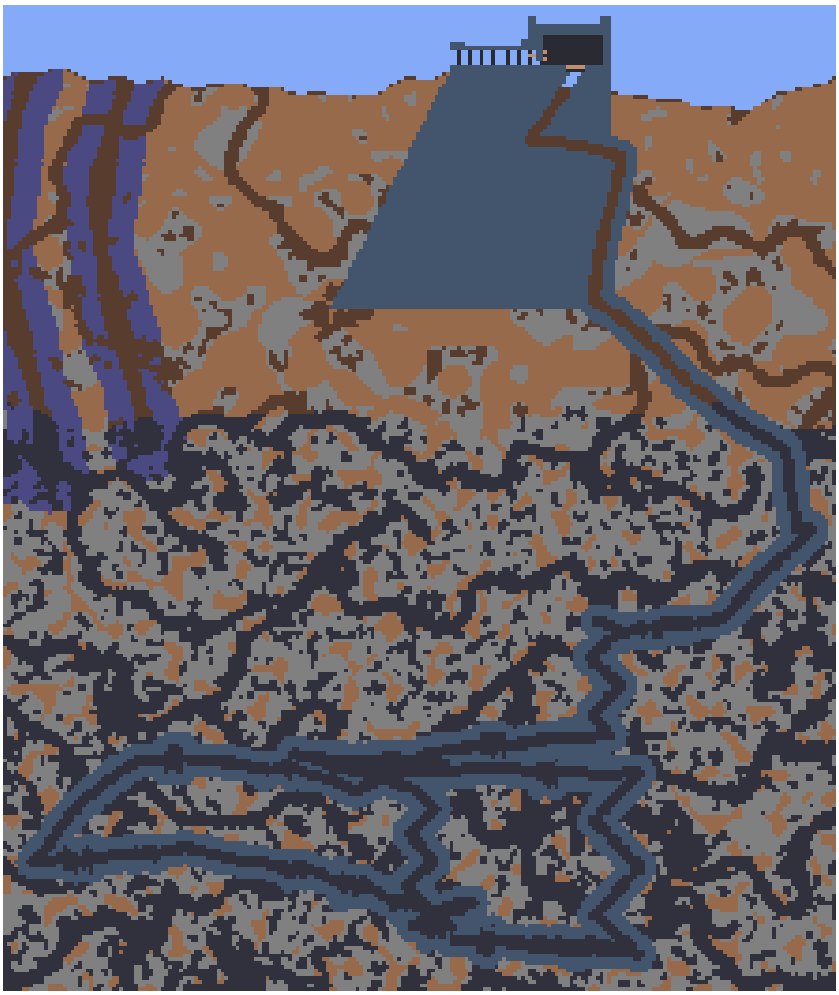
As transições entre estados são estocásticas, garantindo que o layout da masmorra seja imprevisível e desafiador para a navegação. A Figura 17 apresenta um dos resultados obtidos ao se gerar a masmorra.

## 4.7 GERAÇÃO DE ESTRUTURAS ARTIFICIAIS

A inserção de estruturas artificiais, como templos e masmorras, exigiu lógica distinta da geração natural. Para criar templos com formas irregulares (cantos chanfrados), utilizou-se um sistema de verificação de pertinência, permitindo validar se um ponto $(x, y)$ pertence ao interior da forma complexa antes da escavação.

Para evitar que as salas internas parecessem retângulos perfeitos e artificiais, aplicou-se o algoritmo de deslocamento do ponto médio nas bordas das paredes. Embora ambos os algoritmos possuam natureza fractal, o ruído de Perlin fundamenta-se na interpolação suave entre gradientes, o que garante a continuidade da derivada e resulta em curvas com aspecto visualmente "ondulado". Em contrapartida, o algoritmo do ponto médio foi aplicado utilizando interpolação linear entre as subdivisões. Essa abordagem gera descontinuidades na derivada em cada vértice, produzindo uma silhueta serrilhada e angular que simula de forma mais convincente a fratura mecânica e o colapso estrutural em alvenaria de pedra.

Figura 17 – Exemplo de geração da masmorra. Região do mapa ilustrada: $[(3366, 413), (3850, 996)]$



Fonte: Elaborado pelo autor. (2025).

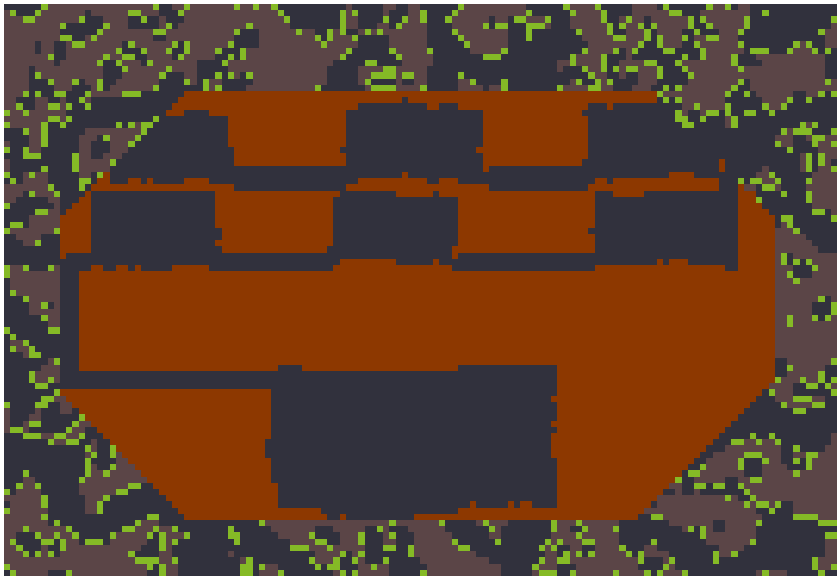
Na implementação do sistema, o algoritmo foi aplicado iterativamente sobre os segmentos de reta que definem os limites das salas, preservando sua estrutura geral, adicionando apenas o "desgaste" visual desejado nas bordas. A Figura 18 demonstra um templo gerado utilizando os métodos citados. A verificação de pertinência é detalhada no Algoritmo 14, enquanto a lógica de refinamento das arestas é apresentada no Algoritmo 15.

## 4.8 ESTRUTURAS LOCALIZADAS

Além dos biomas principais que definem grandes regiões do mapa, o sistema contempla estruturas menores e localizadas que possuem regras de geração específicas, independentes do ruído de terreno global.

Os **Oceanos e Praias** são gerados obrigatoriamente nas extremidades laterais do mundo. Para garantir um declive natural e suave da areia em direção ao fundo do mar,

Figura 18 – Exemplo de geração do Templo da Selva. Região do mapa ilustrada: $[(896, 308), (1191, 510)]$



Fonte: Elaborado pelo autor. (2025).

**Algoritmo 14:** Verificação de pertinência ao formato do templo.

**Input:** Ponto $P(x, y)$, Retângulo $R$, cortes dos cantos $(c_{topo\_esq}, c_{topo\_dir}, \dots)$
**Output:** Verdadeiro se $P$ está dentro do templo, Falso caso contrário

**Passo 1**: Calcule as distâncias para as bordas do retângulo
$dist_{esq} \leftarrow P.x - R.xMin$;
$dist_{dir} \leftarrow R.xMax - P.x$;
$dist_{inf} \leftarrow P.y - R.yMin$;
$dist_{sup} \leftarrow R.yMax - P.y$;

**Passo 2**: Verifique os cortes dos cantos (Chanfro)
**se** $(dist_{esq} + dist_{sup}) < c_{topo\_esq}$ **então**
  **retorna** Falso;
**fim se**
**se** $(dist_{dir} + dist_{sup}) < c_{topo\_dir}$ **então**
  **retorna** Falso;
**fim se**
**se** $(dist_{esq} + dist_{inf}) < c_{base\_esq}$ **então**
  **retorna** Falso;
**fim se**
**se** $(dist_{dir} + dist_{inf}) < c_{base\_dir}$ **então**
  **retorna** Falso;
**fim se**

**Retorne** Verdadeiro;

**Algoritmo 15:** Refinamento de arestas via deslocamento do ponto médio.

**Input:** Lista de segmentos de reta $L$, rugosidade $R \in (0, 1)$, iterações $n$
**Output:** Lista de pontos $P$ representando a parede quebrada

**Passo 1**: Inicialize a lista $P$ com os vértices iniciais da parede;
**Passo 2**: Para cada iteração, subdivida os segmentos existentes
**para** $i \leftarrow 1$ *até* $n$ **faça**
    $NovaLista \leftarrow \emptyset$;
    $Amplitude \leftarrow R^i$ /* *Decaimento da magnitude* */;
    **para cada** *segmento* $(P_a, P_b)$ *em* $P$ **faça**
        $P_{medio} \leftarrow (P_a + P_b)/2$;
        `/* Desloca perpendicularmente ao segmento */`
        $Deslocamento \leftarrow Normal(P_a, P_b) \times Aleatorio(-Amplitude, Amplitude)$;
        $P_{novo} \leftarrow P_{medio} + Deslocamento$;
        Adicione $P_a, P_{novo}$ em $NovaLista$;
    **fim para cada**
    Adicione o ponto final da parede em $NovaLista$;
    $P \leftarrow NovaLista$;
**fim para**
**Retorne** $P$;

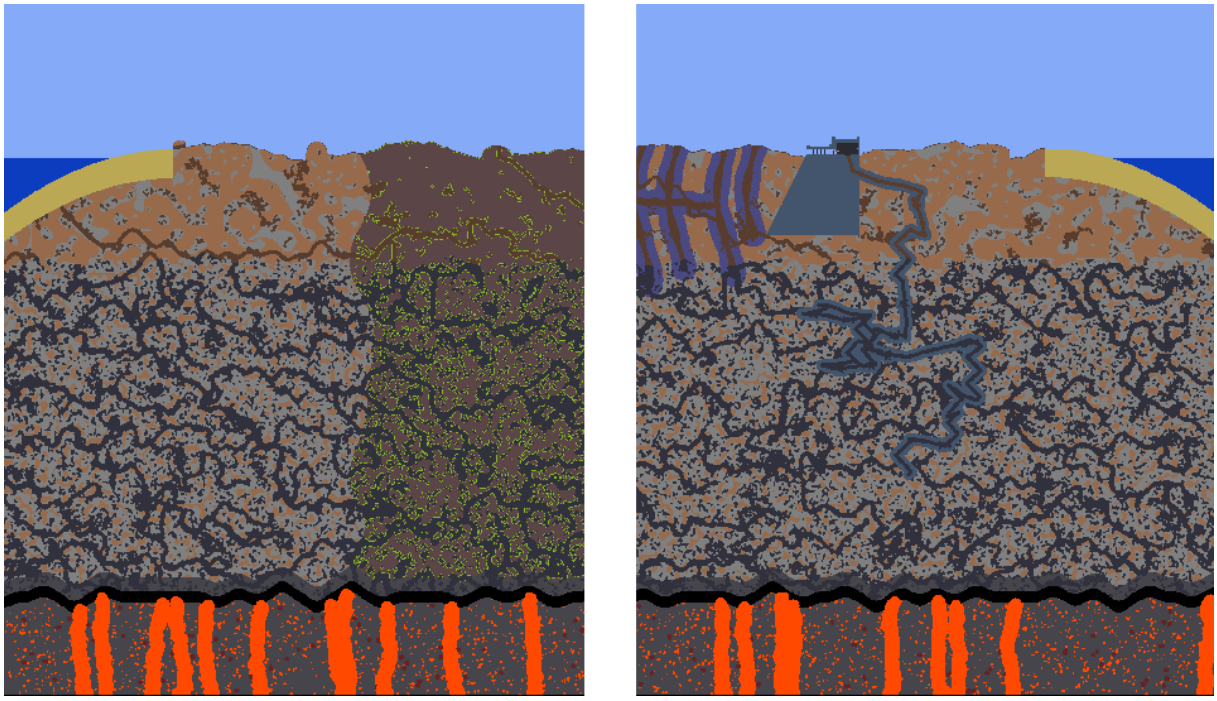
não se utilizou ruído aleatório, mas sim uma função paramétrica. O algoritmo calcula a altura do terreno baseando-se em uma curva de aceleração quadrática ($t^2$) relativa à distância da borda do mapa. Isso cria uma bacia côncava perfeita. Após a definição do relevo, o sistema preenche a cavidade com blocos de água até o nível do mar e substitui a camada geológica imediata por areia, garantindo que o fundo do oceano tenha a espessura correta de sedimento. A Figura 19 demonstra o perfil da praia gerada.

Já no interior da Selva Subterrânea, encontram-se as **Colmeias de Abelha**. Estas estruturas destacam-se por suas formas orgânicas e arredondadas, difíceis de obter com escavadores retangulares tradicionais. O processo de geração, implementado no método `GenerateHive`, ocorre da seguinte forma: primeiramente, um agente escavador cria um túnel irregular no meio da lama; em seguida, cria-se uma borda espessa de blocos de colmeia ao seu redor; por fim, uma porcentagem aleatória inferior da estrutura oca, limitada entre dois valores arbitrários, é preenchida com o bloco líquido de mel ao se transformar os blocos vazios abaixo da altura calculada. O resultado é uma câmara ovalada e natural, como observado na Figura 20.

## 4.9 PREVENÇÃO DE SOBREPOSIÇÃO DE ESTRUTURAS

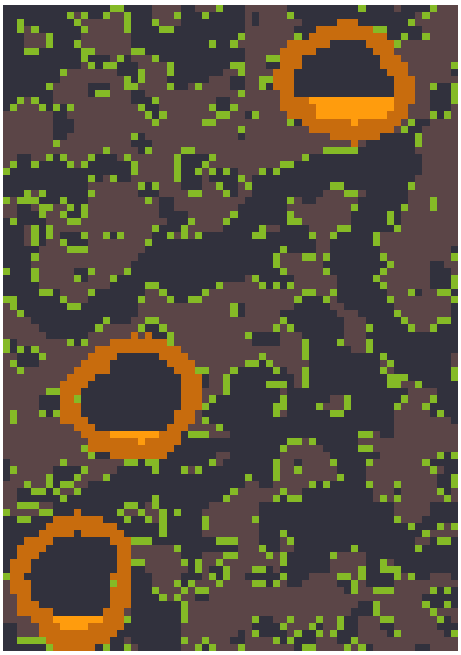
Durante os testes iniciais, notou-se que as cavernas da camada de terra e outras estruturas geradas aleatoriamente se sobrepunham frequentemente, comprometendo a integridade do terreno. A solução adotada foi um sistema de reserva de espaço uti-

Figura 19 – Exemplo de geração do Oceano com declive quadrático. Região do mapa ilustrada: Esquerda: $[(0, 0), (1007, 1200)]$; Direita: $[(3193, 0), (4200, 1200)]$



Fonte: Elaborado pelo autor. (2025).

Figura 20 – Exemplo de geração de uma colmeia. Região do mapa ilustrada: $[(1169, 327), (1312, 530)]$



Fonte: Elaborado pelo autor. (2025).

lizando estruturas `RectInt` para marcar áreas ocupadas. Antes de gerar uma nova caverna, o algoritmo verifica se a área proposta colide com qualquer retângulo na lista `placedSurfaceCaves`.

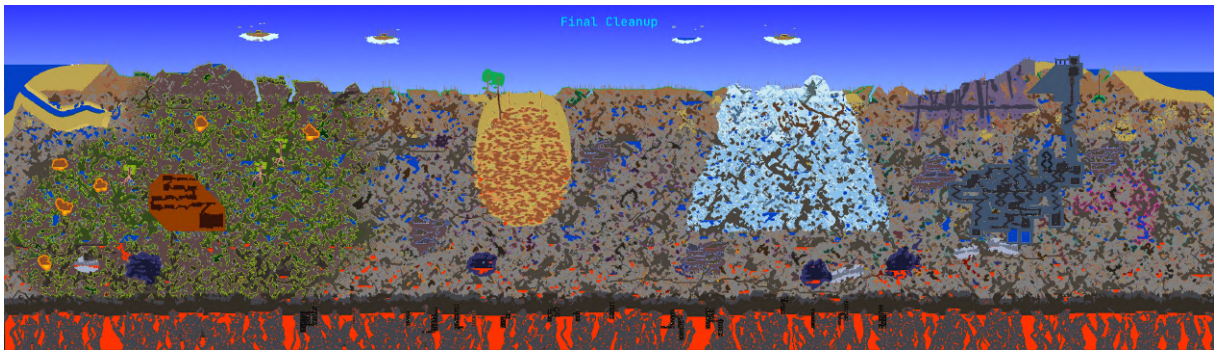
O mesmo princípio foi estendido para montes gerados na superfície, colmeias e ruínas. Para cada tipo de estrutura, dimensões de espaçamento apropriadas foram definidas empiricamente, garantindo que o mapa mantenha uma densidade de pontos de interesse equilibrada sem aglomeração visual.

# 5 RESULTADOS

Este capítulo apresenta a validação visual e a análise dos resultados obtidos através da execução do sistema de geração procedural desenvolvido. O objetivo central deste trabalho foi o estudo de técnicas e algoritmos de geração procedural em mapas bidimensionais, tendo como um estudo de caso as características geológicas e estruturais do jogo Terraria (Re-Logic, 2011). Portanto, a análise a seguir compara diretamente os resultados gerados pelo algoritmo proposto com capturas de tela da geração de mapa do jogo original.

A avaliação foi realizada isolando cada etapa do processo construtivo, permitindo verificar a eficácia dos algoritmos na replicação dos padrões do jogo alvo. A Figura 21 apresenta um exemplo de um mapa do Terraria após todas suas etapas de geração, que serve de base de comparação para os resultados a seguir.

Figura 21 – Exemplo de um mapa completo do Terraria.




Fonte: Re-Logic (2011).

## 5.1 TOPOGRAFIA E TERRENO DE SUPERFÍCIE

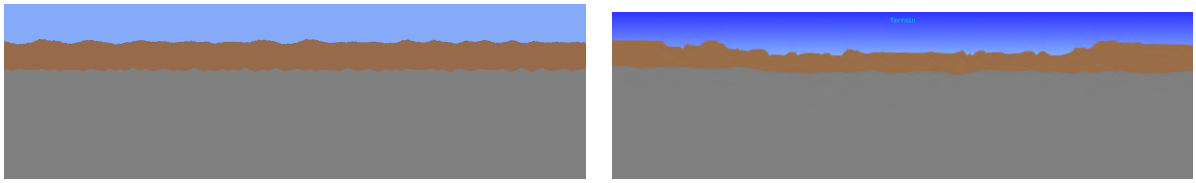
A geração da superfície no Terraria é caracterizada por colinas suaves, ocasionalmente interrompidas por falésias ou pequenas cavernas de entrada, com uma transição clara entre a camada de terra e a camada de pedra. O uso do ruído de Perlin fractal, combinando frequências baixas para o relevo macro e frequências altas para a rugosidade superficial, resultou em uma silhueta que emula as colinas clássicas do jogo. Adicionalmente, a transição entre terra e pedra foi suavizada por um ruído de fronteira, evitando linhas retas artificiais, assemelhando-se à geração orgânica da referência.

O terreno de superfície gerado (camada de terra) utilizou uma altura base $h = 900px$, com o $h = 0$ sendo na parte inferior da figura, e com uma profundidade de $150px$. Para o ruído fractal, foram utilizados os seguintes valores:

- Camada 1: frequência = 0.006 e amplitude = 50;
- Camada 2: frequência = 0.01 e amplitude = 25;
- Camada 3: frequência = 0.05 e amplitude = 10.

A Figura 22 apresenta a comparação entre a superfície gerada pelo algoritmo e a do jogo original. Percebe-se que apesar do Terraria possuir uma variação maior, com alguns relevos mais altos, foi possível obter uma boa aproximação.

Figura 22 – Comparação da topografia de superfície e transição de materiais.



(a) Algoritmo Proposto: Superfície com variação de relevo.

(b) Referência (Terraria): Geração inicial do terreno do Terraria.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

## 5.2 MANCHAS DE AREIA

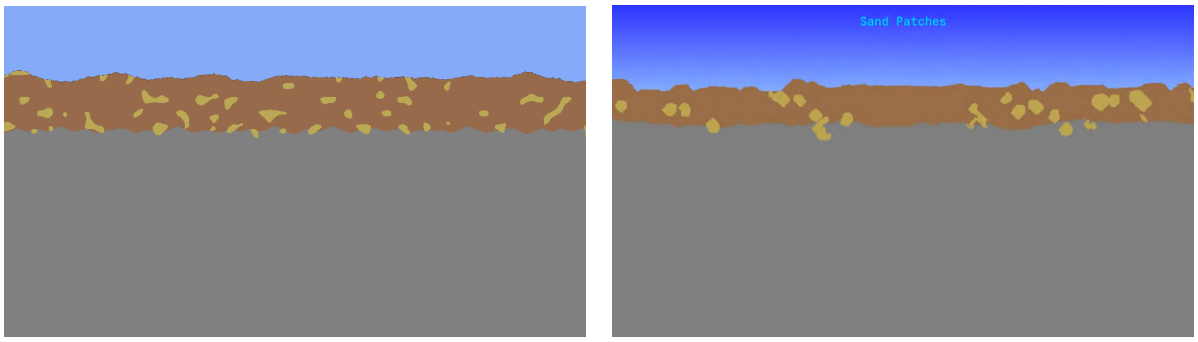
Após a definição da topografia principal, foi executada a etapa `SandPatches` para introduzir heterogeneidade visual e material à superfície. Diferente da geração de biomas, que altera vastas regiões, esta etapa visou criar pequenos depósitos localizados de areia. O algoritmo utilizou a amostragem de Ruído de Perlin (`sandPatchNoiseScale`) descrita na Seção 4.3.

As váriáveis utilizadas para o ruído foram a escala = 0.08 e o limiar = 70%. O resultado foi uma distribuição natural de sedimentos superficiais, conforme observado na Figura 23. Observa-se que o mapa do jogo original apresenta manchas um pouco maiores e, algumas vezes, mais agrupadas, deixando certas áreas bem menos preenchidas que outras.

## 5.3 BOLSÕES DE PEDRA

Para a geração dos bolsões de pedra, foi utilizado, na parte superior do ruído, uma frequência de 0.08 e um limiar de 70%, enquanto para a parte mais profunda utilizou-se uma frequência de 0.04 e um limiar de 55%. Como demonstrado na Figura 24, o algoritmo proposto gerou uma área mais interconectada, lembrando “veios” ou um labirinto, enquanto o Terraria apresenta padrões mais isolados. No entanto, ao se analisar o mapa como um todo, essa diferença se torna quase imperceptível. Mesmo assim, a aproximação conseguiu reproduzir bem a redução da presença de pedras ao se aproximar da superfície.

Figura 23 – Comparação das manchas de areia. Região do mapa ilustrada: $[(1050, 0), (3150, 1200)]$

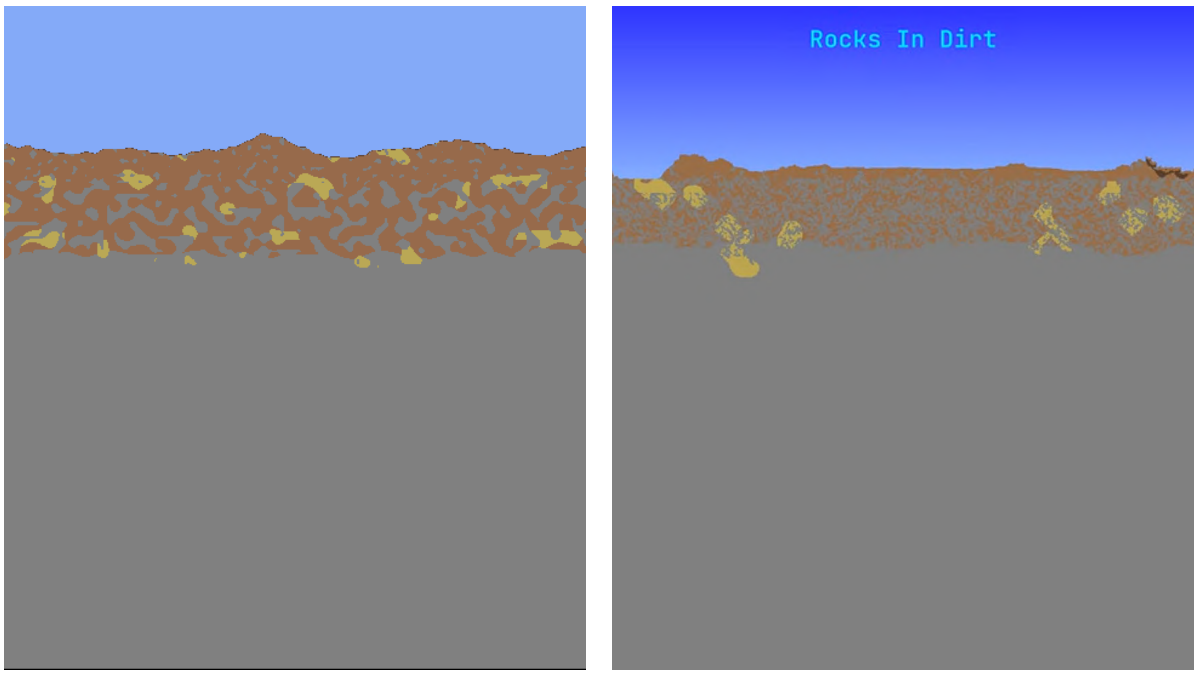



(a) Algoritmo Proposto: Aplicação de manchas de areia.

(b) Referência (Terraria): Adição de manchas de areia.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

Figura 24 – Comparação das manchas de pedra na terra. Região do mapa ilustrada: $[(1050, 0), (3150, 1200)]$




(a) Algoritmo Proposto: Distribuição de pedras na terra.

(b) Referência (Terraria): Adição de pedras na terra.

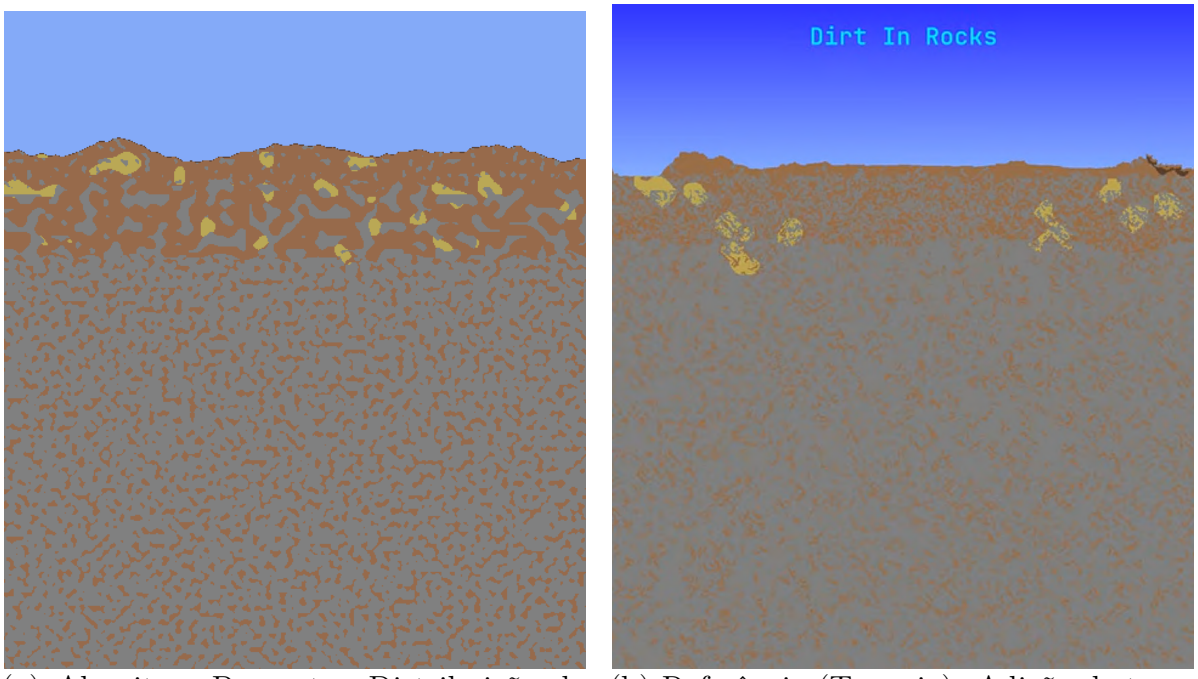
Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

## 5.4 BOLSÕES DE TERRA

Da mesma forma que a camada de terra recebeu intrusões rochosas, a camada de pedra foi processada para incluir depósitos de terra. Esta etapa, executada pelo método `GenerateDirtPatchesInStone`, visa replicar a estratificação imperfeita encontrada na geologia real e no jogo de referência.

A escala utilizada foi de 0.05 e o limiar foi de 62%. A escolha de um limiar mais alto garante que a terra apareça apenas nos “picos” do ruído, resultando em manchas isoladas e orgânicas, em vez de uma mistura homogênea que descaracterizaria a rigidez do subsolo. O resultado visual, conforme ilustrado na Figura 25, cria um contraste entre o marrom da terra e o cinza da pedra. O mapa gerado conseguiu se aproximar muito do mapa do jogo, com manchas mais escassas e bem distribuídas, apesar de o Terraria apresentar manchas com uma aleatoriedade maior no seu agrupamento ao longo do mapa.

Figura 25 – Comparação das manchas de terra na pedra. Região do mapa ilustrada: $[(1050, 0), (3150, 1200)]$




(a) Algoritmo Proposto: Distribuição de terra na pedra.

(b) Referência (Terraria): Adição de terra na pedra.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

## 5.5 ESTRUTURA DE CAVERNAS E SUBSOLO

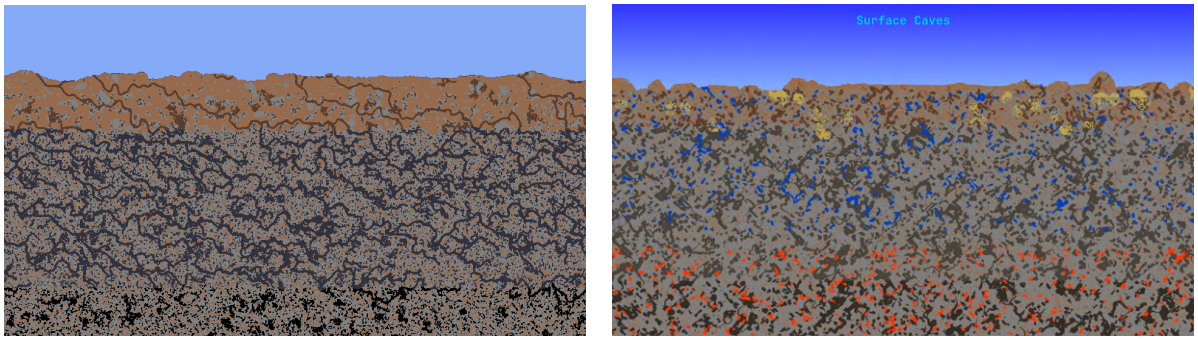
O subsolo do Terraria é dividido em duas zonas principais: a camada de terra, rica em pequenos túneis sinuosos, e a camada de cavernas profundas, caracterizada por grandes salões abertos.

Assim, foram utilizados nas cavernas mais superficiais 600 *walkers* na superfície, com vida útil de 2000 passsos, raio variando entre $2px$ e $3px$ e ondulação $= 0.03$ (varíavel utilizada para obter a direção através do Ruído de Perlin, como descrito na Seção 4.4). Já na camada mais profunda, como parâmetros do autômato celular, utilizou-se uma probabilidade inicial da grade de 46% e foram executadas 5 iterações. 400 *walkers* foram usados para conectar os salões gerados, com vida útil de $1000px$ e persistência de 15%

(chance de se manter na mesma direção a cada passo).

A Figura 26 ilustra a densidade e o formato das cavernas em ambos os casos. É possível notar que o Terraria possui salões mais abertos, e, no caso do algoritmo utilizado, ficaram muito perceptíveis os caminhos percorridos pelos *walkers* tanto na superície quanto na área mais profunda. Apesar disso, as cavernas geradas remetem muito às do jogo.

Figura 26 – Comparação dos sistemas de cavernas subterrâneas. Região do mapa ilustrada: $[(1050, 0), (3150, 1200)]$




(a) Algoritmo Proposto: Túneis sinuosos superiores e salões celulares inferiores.

(b) Referência (Terraria): Visualização do mapa mostrando a densidade de cavernas.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

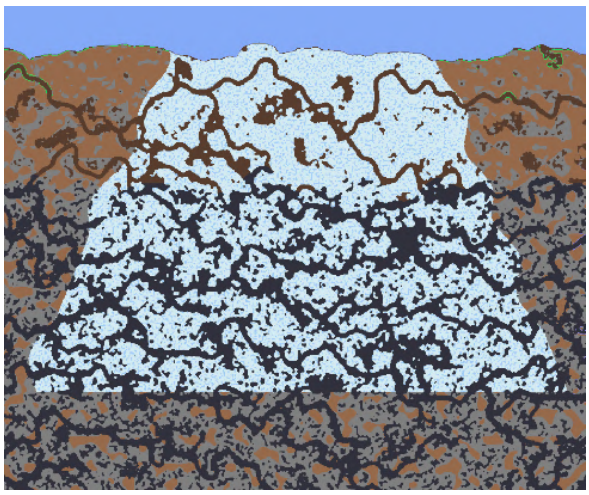
## 5.6 O BIOMA DE NEVE

O bioma de neve no Terraria possui uma característica topológica muito específica: ele funciona como um funil ou trapézio invertido, sendo mais estreito na superfície e expandindo-se horizontalmente conforme a profundidade aumenta, dominando grande parte do subsolo do lado oposto à selva. Assim, o bioma foi gerado com uma largura na base superior de $400px$ e na base inferior de $800px$, e com uma profundidade de $500px$. Além disso, para o Ruído de Perlin 1D aplicado nas suas bordas, foi utilizado uma frequência de 0.01 e amplitude de 50. A Figura 27 compara o formato do bioma gerado com o do jogo. Observa-se que foi possível se aproximar bem do formato do bioma original, tanto em uma visão geral quanto em sua fronteira.

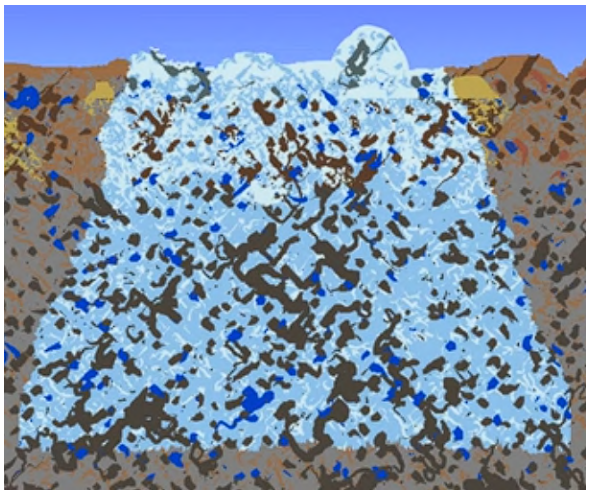
## 5.7 O BIOMA DE SELVA

A Selva Subterrânea representa um aumento significativo na complexidade da geração, pois envolve não apenas a alteração da topologia, mas a conversão total dos materiais de uma vasta região e a inserção de sub-estruturas líquidas e arquitetônicas. A geração deste bioma foi dividida em três etapas distintas para garantir a correta sobreposição dos elementos: a definição do bioma, a formação das colmeias e a construção do templo.

Figura 27 – Comparação da morfologia do Bioma de Neve. Região do mapa ilustrada: $[(2615, 270), (3407, 1033)]$



(a) Algoritmo Proposto: Formato trapezoidal com manchas de gelo.



(b) Referência (Terraria): Bioma de neve subterrâneo.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

Para a geração do bioma, utilizou-se uma largura de $800px$ e uma profundidade que ia até o encontro com o limite do submundo. Já para a suavização de suas bordas com o Ruído de Perlin 1D, utilizou-se uma frequência de 0.008 e amplitude de 50.

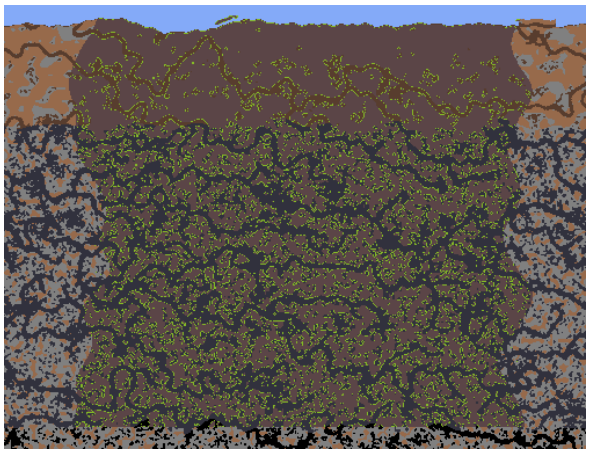
A primeira etapa consistiu na delimitação da área da selva e na conversão dos materiais geológicos. Diferente dos outros biomas que apenas substituem blocos, a selva exige que a terra e pedras sejam convertidas em lama. O algoritmo percorreu a região designada aplicando uma verificação de vizinhança: blocos de pedra adjacentes a cavernas abertas foram convertidos em grama de selva (*Jungle Grass*), enquanto a pedra maciça foi transformada em lama. Isso preservou o formato das cavernas geradas nas etapas anteriores, apenas alterando sua representação visual, como observado na Figura 28. Percebe-se que, no exemplo do Terraria, a selva apresenta um formato mais complexo, com uma parte que se expande mais lateralmente, mas essas diferenças são por conta do encontro com outro bioma (nesse caso, o do Oceano), fato que não aconteceu na geração do algoritmo desenvolvido.

## 5.8 O DESERTO

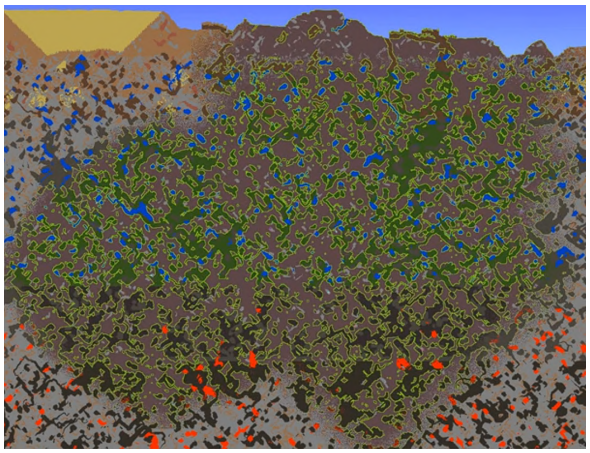
No jogo de referência, o deserto subterrâneo possui um formato oval distinto e cavernas que lembram tocas de formigas, com paredes revestidas. Para sua geração, foi utilizado o valor de $400px$ para a largura e $500px$ para a altura. Além disso, para o ruído de Pelin *ridged*, a frequência foi definida como 1, as oitavas como 3 e a amplitude como 1.

Diferente do ruído de Perlin comum que cria formas suaves, o ruído *ridged* cria

Figura 28 – Comparação da composição material da Selva. Região do mapa ilustrada: $[(460, 136), (1559, 980)]$



(a) Algoritmo Proposto: Geração incial da área da selva.



(b) Referência (Terraria): Densidade de vegetação e lama.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

"estrias" e túneis finos e caóticos. A aplicação automática de paredes de arenito (`Sandstone`) ao redor desses túneis completou a estética visual, conforme comparado na Figura 29. A análise revela que o formato elíptico de ambos os biomas é muito próximo, além de as cavernas serem bastante semelhantes, apesar das cavernas do jogo aparentarem ser menores e mais caóticas do que no algoritmo gerado.

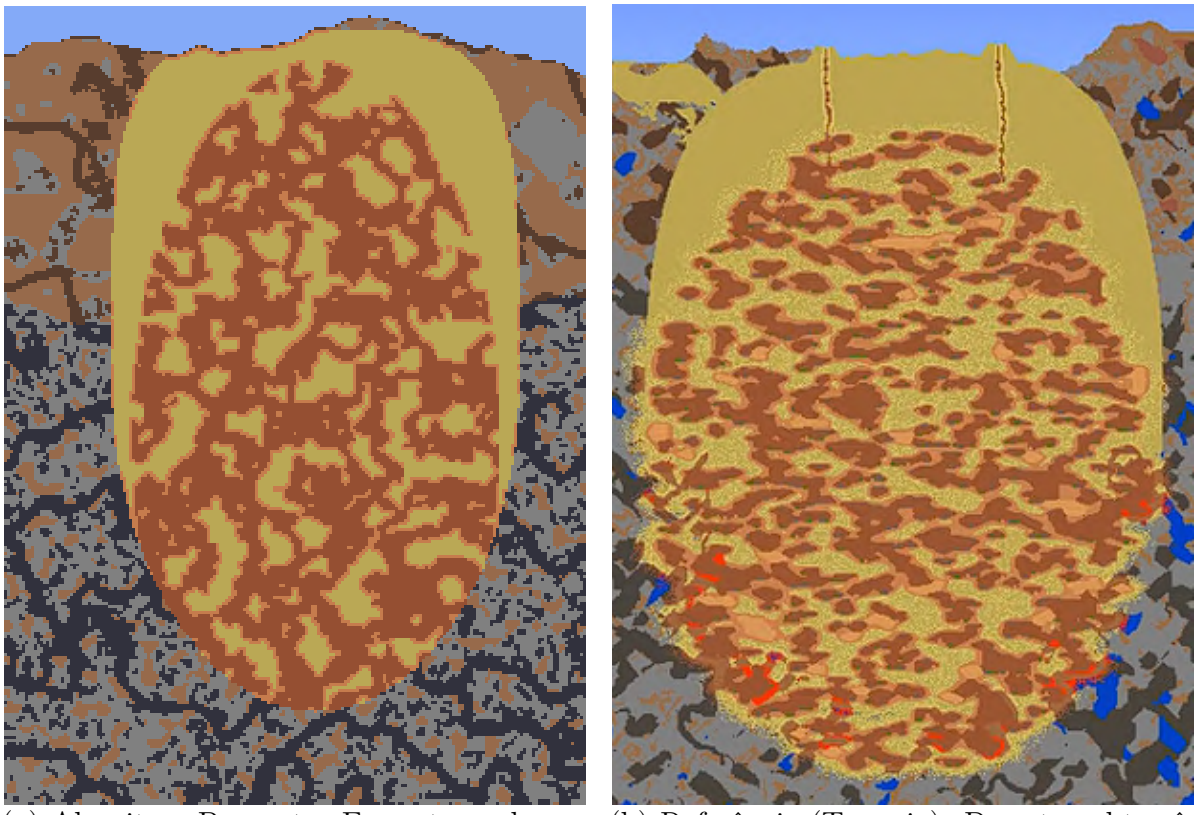
## 5.9 O SUBMUNDO

O Submundo é caracterizado no jogo original por um teto plano de cinzas, grandes lagos de lava e estruturas em ruínas. Para gerá-lo, foi utilizada uma altura de $180px$, com o espaço entre o teto e o chão de $20px$. Foram utilizados 200 *walkers* para a caminhada aleatória dos rios de lava, com raio variando entre $3px$ e $6px$, um espaçamento mínimo entre um rio e outro de $10px$ e uma chance de 60% de mover numa direção que não seja para baixo. Já para o autômato que gerou os poços de lava, foi utilizada uma porcentagem para a geração da malha inicial de 4% e 2 iterações. O resultado pode ser observado na Figura 30.

## 5.10 O BIOMA DE CORRUPÇÃO

Uma das características mais icônicas do estudo de caso é o bioma de "Corrupção", definido por abismos verticais profundos que descem da superfície até o subsolo, conectados por um túnel horizontal no fundo. O algoritmo implementado replicou essa estrutura através de 5 escavadores verticais cuja trajetória horizontal foi modulada por uma função seno de frequência 0.02, criando as paredes onduladas características. Sua profundidade

Figura 29 – Comparação da morfologia do Deserto Subterrâneo. Região do mapa ilustrada: $[(1774, 380), (2194, 971)]$



(a) Algoritmo Proposto: Formato oval com cavernas estriadas.

(b) Referência (Terraria): Deserto subterrâneo e seus túneis característicos.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

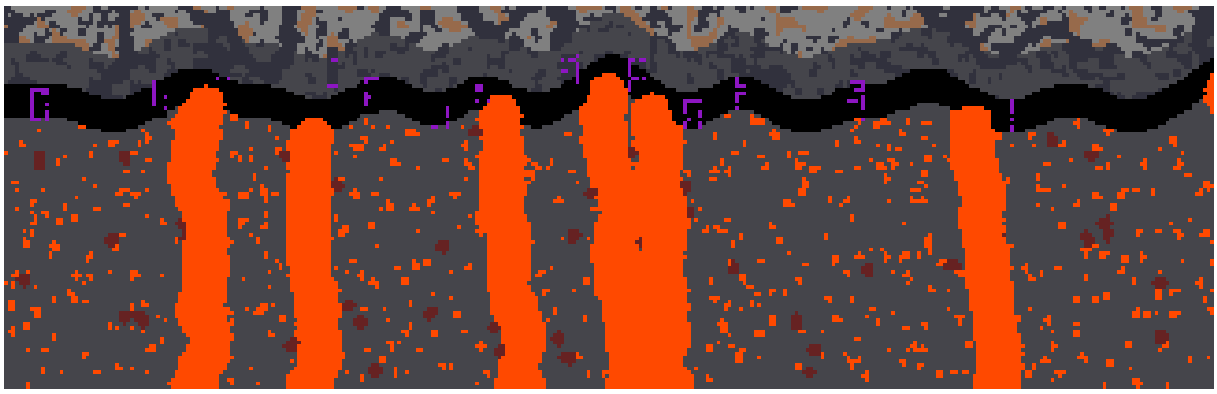
pôde variar entre $150px$ e $350px$, e suas larguras entre $4px$ e $8px$, com um espaçamento de no mínimo $30px$ entre cada escavador. O túnel horizontal foi gerado com tamanho de $400px$ e raio de $6px$. O bioma tinha largura máxima de $500px$ e foi preenchido com pedra corrompida e a grama da superfície alterada. Como observado na Figura 31, os túneis gerados assemelham-se significativamente aos túneis do jogo, apesar de estarem em menor quantidade.

## 5.11 A MASMORRA (DUNGEON)

A geração da Masmorra combinou a inserção de um modelo pré-fabricado (*Prefab*) para a entrada, garantindo que a estrutura na superfície seja reconhecível e esteticamente agradável, com a geração totalmente procedural do interior. O diâmetro utilizado para o buraco de entrada que inicia a masmorra foi de $11px$. Foi utilizado apenas um *walker* que se ramifica posteriormente, com vida útil de $1500px$.

O modo de labirinto inicia aleatoriamente entre $250px$ e $350px$ de profundidade. Nele, o agente tem chance de 1% de se dividir em dois, com o filho tendo vida útil de

Figura 30 – Comparação do ambiente do Submundo. Região do mapa ilustrada: $[(1796, 0), (2491, 237)]$

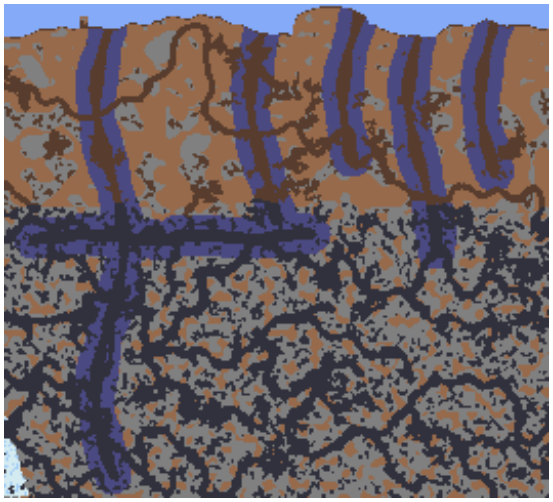


(a) Algoritmo Proposto: Lagos de lava e ruínas no bioma de cinzas.

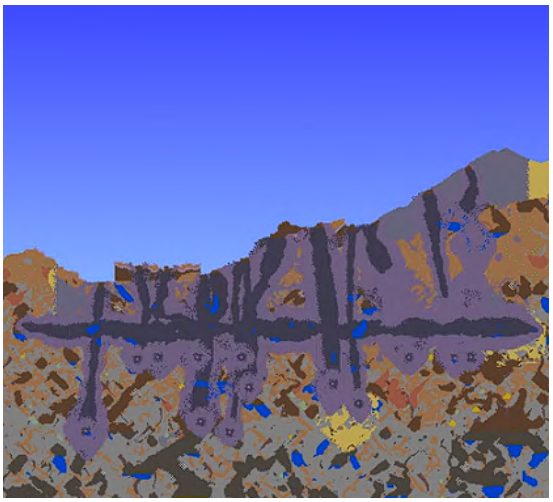
(b) Referência (Terraria): O Submundo com suas torres e lava.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

Figura 31 – Comparação estrutural do bioma de Corrupção. Região do mapa ilustrada: $[(3330, 435), (3915, 958)]$



(a) Algoritmo Proposto: Abismos verticais conectados por túnel horizontal.



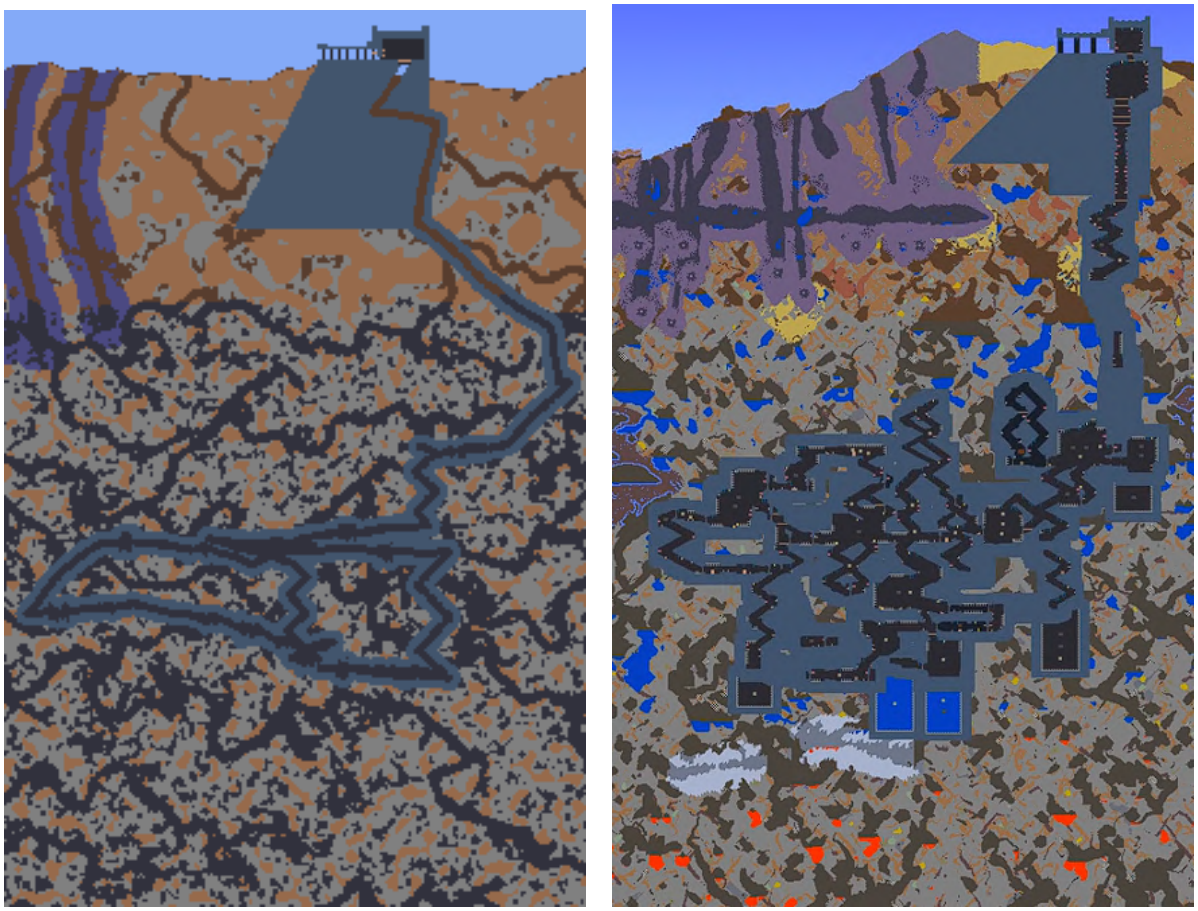
(b) Referência (Terraria): Estrutura clássica dos abismos da Corrupção.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

600$px$. O *walker* tem ainda 0.5% de chance de entrar no modo Zig-Zag. Uma vez neste modo, deve fazer no mínimo 2 e no máximo 5 zig-zags, com tamanho mínimo de 20$px$ e máximo de 40$px$.

A comparação na Figura 32 demonstra que, apesar de não ter conseguido gerar as salas como as do Terraria, que foi uma limitação do algoritmo desenvolvido, o caminho gerado pelos túneis se assemelha ao original, tanto na descida quanto no labirinto que é gerado profundamente.

Figura 32 – Comparação da arquitetura da Masmorra. Região do mapa ilustrada: $[(3361, 242), (3852, 998)]$



(a) Algoritmo Proposto: Entrada pré-fabricada e labirinto de tijolos.

(b) Referência (Terraria): Entrada da Dungeon e layout interno.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

## 5.12 OCEANOS E PRAIAS

Os oceanos no Terraria localizam-se nas extremidades laterais do mapa, caracterizados por uma descida íngreme de areia que forma uma bacia profunda preenchida por água. Para reproduzir esse declive suave, a bacia resultante foi preenchida numa largura de 300$px$ com água com uma profundidade de 100$px$ e o subsolo imediato convertido para

areia numa profundidade de $50px$, garantindo que o fundo do mar tivesse a espessura correta, sem expor a pedra ou terra subjacente de forma abrupta. A Figura 33 demonstra a similaridade na curvatura da praia e no preenchimento do volume de água, apesar do oceano do Terraria possuir algumas variações maiores no seu formato, não se baseando somente na curva quadrática.

## 5.13 O TEMPLO DA SELVA

Em contraste com o caos orgânico da lama e das colmeias, o Templo da Selva é uma estrutura artificial rígida. Sua geração foi um desafio de geometria procedural e conectividade das salas.

A forma externa foi gerada com uma largura de $120px$ e altura de $100px$, com os cortes podendo variar entre $20px$ e $40px$. Os corredores internos utilizaram um raio de $3px$, com suas larguras antes de gerar a próxima sala entre $30px$ e $50px$. A largura das salas variam entre $12px$ e $22px$, e a altura entre $10px$ e $16px$. Já a última sala, que é maior, tem sua largura fixa em $60px$ e altura em $30px$. Para o algoritmo de deslocamento de ponto médio, utilizado para dar uma sensação de serrilhado nas bordas das salas, foi utilizado uma amplitude de 3 e uma rugosidade de 0.6.

Na Figura 34, é possível verificar que as salas do Terraria apresentam melhor a aparência de ruínas e os corredores tem mais variações em suas direções. Todavia, foi possível criar um templo com salas conectadas e com as mesmas características básicas das do jogo.

## 5.14 AS COLMEIAS DE ABELHA

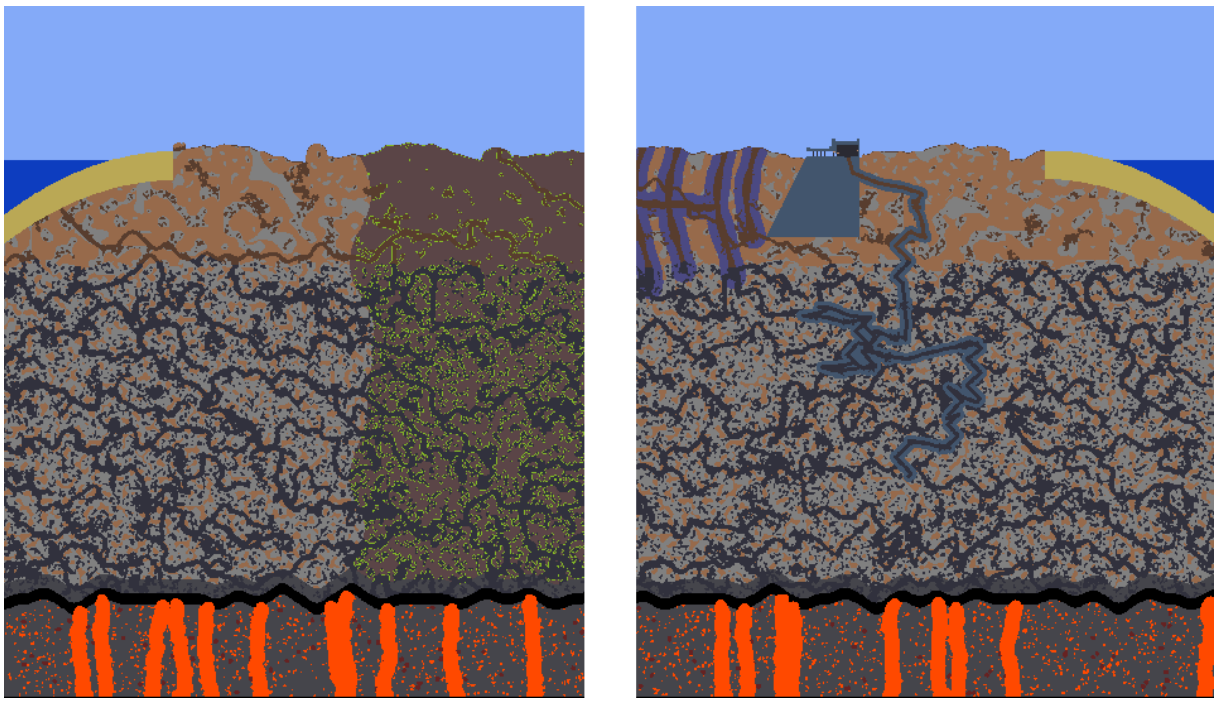
Uma das estruturas mais características do bioma da selva são as colmeias gigantes. Para replicar sua forma orgânica e arredondada, utilizou-se a técnica de caminhada aleatória inicialmente, com *walkers* com vida de $150px$ e um raio de $5px$. Após isso, o buraco gerado é expandido em suas bordas com blocos de colméia numa espessura de $5px$, finalizando-se com o preenchimento da parte inferior da estrutura com o bloco líquido de mel numa porcentagem de ocupação do buraco entre 10% e 30%.

A Figura 35 demonstra como essa abordagem gerou estruturas que parecem ter crescido organicamente, evitando formas geométricas rígidas, apesar das colméias do jogo possuírem um formato mais ondulado e variado. Além disso, foi possivel reproduzir fielmente o mel presente dentro dessas colméias.

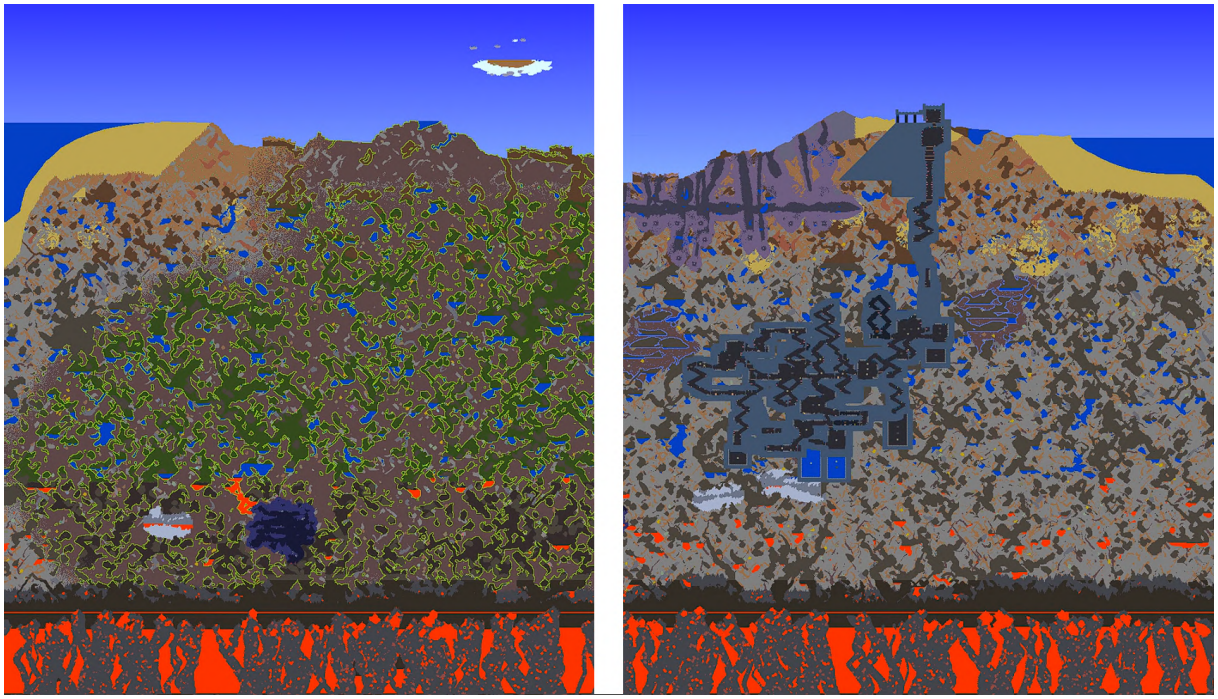
## 5.15 ANÁLISE DO MAPA COMPLETO

Para concluir a validação dos resultados, foi realizada uma análise macroscópica do mapa inteiro gerado pelo algoritmo, comparando-o com a visão global de um mundo

Figura 33 – Comparação da geração dos Oceanos. Região do mapa ilustrada: Esquerda: $[(0, 0), (1007, 1200)]$; Direita: $[(3193, 0), (4200, 1200)]$



(a) Algoritmo Proposto: Declive quadrático preenchido com água.



(b) Referência (Terraria): Oceano e limite lateral do mundo.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

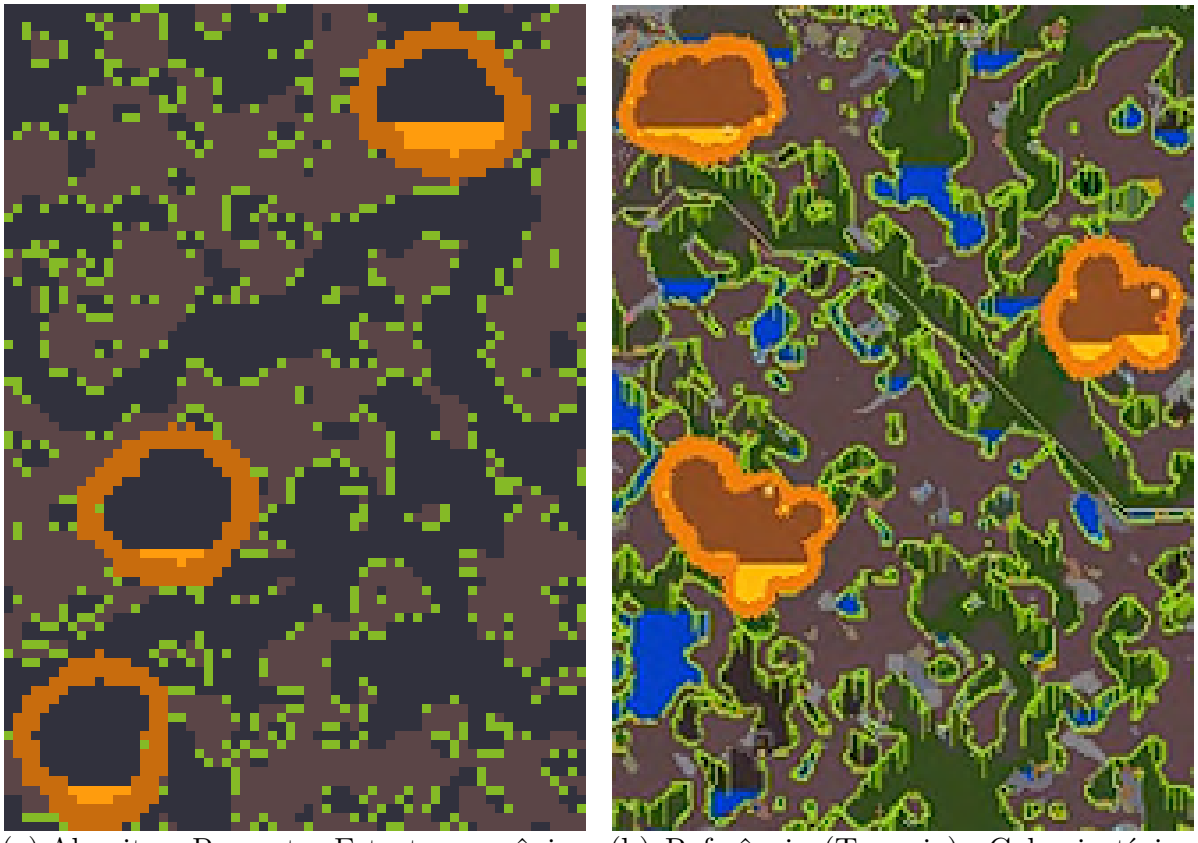
Figura 34 – Comparação da arquitetura do Templo da Selva. Região do mapa ilustrada: $[(896, 308), (1191, 510)]$

(a) Algoritmo Proposto: Estrutura de tijolos com rota interna conectada.

(b) Referência (Terraria): O Templo Lihzahrd.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

Figura 35 – Comparação das Colmeias de Abelha. Região do mapa ilustrada: $[(1169, 327), (1312, 530)]$



(a) Algoritmo Proposto: Estrutura orgânica com mel líquido.

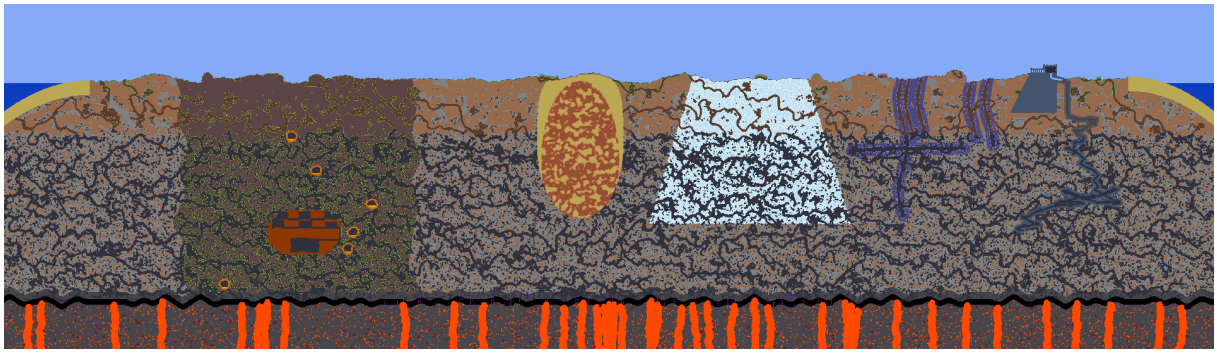
(b) Referência (Terraria): Colmeia típica com larva e mel.

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

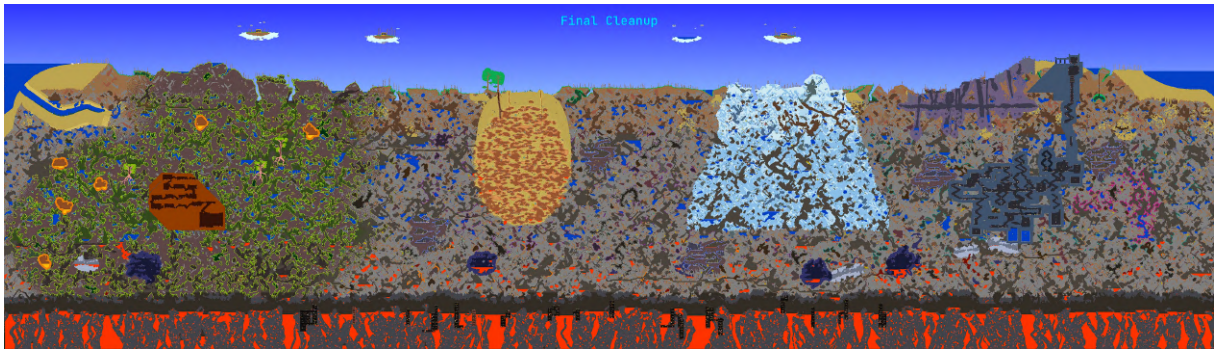
gerado no Terraria. Esta comparação visa verificar não apenas a presença dos elementos individuais discutidos anteriormente, mas a distribuição lógica e a coerência topológica do mundo como um todo.

A Figura 36 apresenta, lado a lado, o resultado final da geração procedural proposta e um mapa de referência.

Figura 36 – Comparação global entre o mapa gerado e o jogo original.



(a) Mapa Completo Gerado (4200x1200 blocos)




(b) Mapa de Referência (Terraria)

Fonte: (a) Elaborado pelo autor (2025); (b) Re-Logic (2011).

A análise da imagem demonstra que o sistema de geração respeitou rigorosamente as camadas de profundidade clássicas do gênero. É possível identificar visualmente a transição da camada de Superfície (com o céu azul e a grama verde) para o Subsolo (terra marrom), descendo para a camada de Cavernas (pedras cinzas) e finalizando no Submundo (cinzas cinza escuro e lava laranja). A proporção dessas camadas foi ajustada parametricamente para coincidir com a experiência de jogo, onde a camada de cavernas ocupa a maior parte da área jogável, oferecendo o maior espaço para exploração.

Embora o mapa de referência na Figura 36b apresente uma complexidade maior em termos de micro-biomas (como granito, mármore e ilhas flutuantes), que fugiram ao escopo deste trabalho, a estrutura fundamental (“esqueleto”) do mundo gerado pelo sistema (Figura 36a) é notavelmente similar.

A silhueta do terreno, a presença de oceanos nas bordas limitantes e a conectividade geral das cavernas mostram que a combinação de algoritmos clássicos como o ruído de Perlin, autômatos celulares e caminhada aleatória é extremamente poderosa para criar a base de um mundo virtual complexo e coerente, muito próximo em sua macro-estrutura de um produto comercial consolidado.

# 6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento e a análise de um sistema de geração procedural de mapas bidimensionais focado em jogos de plataforma e exploração, utilizando o jogo Terraria como estudo de caso principal. O objetivo central de criar ambientes variados, estruturalmente coerentes e esteticamente próximos à referência foi alcançado através da orquestração de múltiplos algoritmos clássicos de GPC e da computação gráfica.

A abordagem híbrida adotada, combinando ruído de Perlin para a macro-topografia, autômatos celulares para erosão subterrânea e caminhada aleatória para escavação de túneis, permitiu a reprodução de várias das características do mapa de referência, como pode ser constatado pelos resultados visuais obtidos. O uso do ruído de Perlin fractal foi eficaz para evitar a repetição monótona do terreno, criando colinas e vales naturais, ao passo que as máscaras de ruído permitiram transições suaves entre materiais, como a mistura orgânica de terra e pedra. A combinação de escavadores e autômatos celulares também foi fundamental para criar um subsolo composto por túneis sinuosos e grandes salões. Também foi possível reproduzir os diversos biomas e estruturas do jogo a partir da combinação dos algoritmos com lógicas específicas para controle do processo de geração. Na geração da Masmorra (*Dungeon*) e do Templo da Selva, a capacidade de alternar entre modos de escavação (labirinto e corredor) permitiu criar estruturas bem semelhantes às do jogo original.

Apesar dos resultados satisfatórios, o sistema desenvolvido apresenta limitações. Para mapas de grandes dimensões, o tempo de geração pode se tornar elevado, impactando o tempo de carregamento inicial. Além disso, embora os algoritmos de Caminhada Aleatória tendam a criar caminhos conectados, o sistema atual não possui uma etapa de validação desses caminhos para garantir que todas as áreas são acessíveis sem a necessidade de o jogador quebrar blocos. Por outro lado, apesar de ser uma prática interessante, o caso utilizado do Terraria também não garante essa conectividade. Por fim, o trabalho focou na geração da estrutura do mundo (blocos e paredes). Elementos dinâmicos como a simulação de fluidos, queda de areia (física de blocos) e a distribuição de vegetação complexa (árvores *multitiles*) foram simplificados ou não abordados.

Como trabalhos futuros, pode-se pensar em adaptá-lo para a geração de partes do mapa sob demanda. Isso poderia melhorar o desempenho do processo de geração, além de permitir a criação de mundos virtualmente infinitos, similar ao funcionamento do jogo Minecraft. Contudo, isso introduz outros desafios relacionados à manutenção da coerência entre os biomas gerados. Outra linha possível de ser seguida é a avaliação da conectividade do mapa, para detectar áreas isoladas e gerar túneis de conexão automaticamente. Além disso, seria possível expandir a geração procedural para outros elementos de jogo, como a inserção de mobiliário, baús de tesouro e vegetação, utilizando regras de contexto (ex:

baús apenas em salas fechadas da masmorra ou em cavernas de mais difícil acesso) para enriquecer a narrativa do ambiente gerado.

# REFERÊNCIAS

SHAKER, N.; TOGELIUS, J.; NELSON, M. **Procedural Content Generation in Games**. Springer, 2016.

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**, v. 3, n. 3, p. 172-186, 2011

LEITE, Gabriel de Oliveira Barbosa; LIMA, Edirlei Soares de. **Geração Procedural de Mapas para Jogos 2D**. SIMPÓSIO BRASILEIRO DE GAMES E ENTRETENIMENTO DIGITAL (SBGAMES), p. 244-247, 14., 2015.

FARUOLO, Carlos Felipe Medeiros; AGUIAR, Felipe Pimentel de. **Geração Procedural de Mapas para Jogos de Plataforma**. Projeto Final de Graduação – Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2014.

FOURNIER, A.; FUSSELL, D.; CARPENTER, L. (1982). **Computer rendering of stochastic models.** *Communications of the ACM*, 25(6), 371-384.

GARDNER, M. **The fantastic combinations of John Conway's new solitaire game "life"**. Scientific American, v. 223, n. 4, p. 120-123, 1970.

Foley, J. D.; Van Dam, A.; Feiner, S. K.; Hughes, J. F. **Computer Graphics: Principles and Practice**. Addison-Wesley Professional, 2nd, 1996.

LAGAE, A.; LEFEBVRE, S.; DRETTAKIS, G.; DUTRÉ, P. **Procedural noise using sparse Gabor convolution**. ACM Transactions on Graphics (TOG), New York, v. 29, n. 4, p. 1-10, 2010.

MATHERON, G. **Random Sets and Integral Geometry**. New York: Wiley, 1975.

PERLIN, K. **An image synthesizer**. ACM SIGGRAPH Computer Graphics, New York, v. 19, n. 3, p. 287-296, 1985.

PERLIN, K. **Improving noise**. ACM Trans. Graph. 21, 3, p. 681-682, 2002.

PAVLIDIS, T. **Algorithms for graphics and image processing**. Rockville: Computer Science Press, 1982.

SERRA, J. **Image Analysis and Mathematical Morphology**. London: Academic Press, 1982.

SERRA, J. **Image analysis and mathematical morphology. Volume 2: Theoretical Advances**. London: Academic Press, 1988.

SPITZER, F. **Principles of Random Walk**. 2. ed. New York: Springer Science & Business Media, 2001.

WOLFRAM, S. **A New Kind of Science**. Champaign: Wolfram Media, Inc., 2002.

DUBSKÝ, Tomáš. **Procedural Generation and Simulation of 2D Gaming World**. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology, 2022.

ROSE, T. J.; BAKAOUKAS, A. G. **Algorithms and Approaches for Procedural Terrain Generation**. Northampton: University of Northampton, School of Science and Technology, 2016.

MACEDO, Y. P. A.; CHAIMOWICZ, L. **Improving procedural 2D map Generation based on multi-layered cellular automata and Hilbert curves.** 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), IEEE, 2017.

GUILHERME NEVES CANEDO. **Criação de Mapas utilizando Geração Procedural**. Pontifícia Universidade Católica de Goiás, 2022.

JOHNSON, L.; YANNAKAKIS, G. N. ; TOGELIUS, J. **Cellular automata for real-time generation of infinite cave levels**. Workshop on Procedural Content Generation in Games, PC Games 2010, Co-located with the 5th International Conference on the Foundations of Digital Games. 2010

EBERT, D. S.; MUSGRAVE, F. K.; PEACHEY, D.; PERLIN, K.; WORLEY, S. **Texturing and Modeling: A Procedural Approach**. The Morgan Kaufmann Series in Computer Graphics. 2002

FISCHER, R.; DITTMANN, P.; WELLER, R.; ZACHMANN, G. **AutoBiomes: procedural generation of multi-biome landscapes**. Springer. The Visual Computer, v. 36, n. 10-12, p. 2227-2236, 2020.

GELLEL, A.; SWEETSER, P. **A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels**. ACM. Proceedings of the 15th International Conference on the Foundations of Digital Games, p. 1-10, 2020.

SALINAS, J. **An Exploration of Procedural Methods in Game Level Design**. Undergraduate Honors Thesis, University of Arkansas, 2023.

SEPANMAA, T. **Procedural Level Generation in 2D Roguelite Games**. Tampere University of Applied Sciences, 2024.

PIISPA, K. **Methods for Procedural Terrain Generation**. University of Turku, 2022.

BABIN, M. P. **Hybrid Approaches to Procedural Content Generation for Game Design, Production, and Security**. The University of Western Ontario, 2025.

Accidental Noise library. **3D Cube World Level Generation.** Disponível em: https://accidentalnoise.sourceforge.net/minecraftworlds.html. Acesso em: 18 jul. 2025.

tModLoader Team. **Vanilla World Generation Steps**. Disponível em: https://github.com/tModLoader/tModLoader/wiki/Vanilla-World-Generation-Steps. Acesso em: 12 jan. 2026.

Unity Community. **Procedural Terrain Generation Algorithms**. Disponível em: https://forum.unity.com/threads/procedural-terrain-generation-algorithms.236442. Acesso em: 12 jan. 2026.