Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Geração procedural de plantas baixas para jogos digitais

Leandro Dornela Ribeiro

JUIZ DE FORA
AGOSTO, 2025

Geração procedural de plantas baixas para jogos digitais

Leandro Dornela Ribeiro

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Marcelo Caniato Renhe

Coorientador: Igor de Oliveira Knop

JUIZ DE FORA

AGOSTO, 2025

Geração procedural de plantas baixas para jogos digitais

Leandro Dornela Ribeiro

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Marcelo Caniato Renhe
Doutor em Engenharia de Sistemas e Computação

Igor de Oliveira Knop
Doutor em Modelagem Computacional

Flávia de Souza Bastos
Doutora em Engenharia Mecânica

Rodrigo Luis de Souza da Silva
Doutor em Engenharia

JUIZ DE FORA
14 DE AGOSTO, 2025

Aos meus amigos e irmãos.

Aos pais, pelo apoio e sustento.

Resumo

A crescente complexidade e escala dos jogos modernos, aliada à necessidade de criação de grandes volumes de conteúdo, torna a geração procedural de conteúdo uma estratégia valiosa para reduzir custos e tempo de desenvolvimento. Este trabalho traz um estudo em geração procedural em jogos digitais com foco em plantas baixas para interiores exploráveis. Inicialmente, foi realizado uma revisão da literatura para encontrar métodos adequados a esse propósito. Foi escolhido e implementado um método que atendesse a requisitos como facilidade de implementação, possibilidades de melhoria, expansão e integração. Foi elaborado um conjunto de casos de teste para avaliação dos resultados obtidos pelo algoritmo. A implementação, na forma de uma ferramenta integrada ao motor de jogos Unity, foi capaz de se aproximar dos resultados do trabalho original, embora os testes sugiram a necessidade de melhorias para reduzir o número de resultados inválidos.

Palavras-chave: jogos digitais, geração procedural de conteúdo, geração de interiores.

Abstract

The increasing complexity and scale of modern games, combined with the need to create large volumes of content, makes procedural content generation a valuable strategy for reducing costs and development time. This work presents a study of procedural content generation in digital games, focusing on floor plans for explorable interiors. Initially, a literature review was conducted for identifying adequate methods for this purpose. The selected method was chosen to meet requirements such as ease of implementation, possibilities for improvement, expansion, and integration. A set of test cases was devised for evaluating the algorithm results. The implementation, in the form of a tool integrated with the Unity game engine, was able to approximate the results found in the original work, even though the experiments indicated the need for improvements in order to reduce the number of invalid results.

Keywords: digital games, procedural content generation, interior generation.

Agradecimentos

A toda minha família e camaradas, pelo encorajamento e apoio.

Aos professores Marcelo Caniato e Igor Knop pela orientação, amizade e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

"A ciência é muito mais que um corpo de conhecimentos. É uma maneira de pensar."

Carl Sagan

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
PCG	Procedural content generation
GPC	Geração procedural de conteúdo
NPC	Non-player character
WPCG	Workshop on Procedural Content Generation

1 Introdução

Desde a década de 80, jogos digitais vêm fazendo uso de métodos para gerar conteúdo devido às limitações de hardware (Shaker et al, 2016). Com a constante demanda por projetos maiores, mais realistas e dinâmicos, a Geração Procedural de Conteúdo (GPC) ganhou espaço e se popularizou na indústria de jogos digitais. Shaker et al (2016) define GPC como um software capaz de criar conteúdo para um jogo de maneira algorítmica e automatizada a partir de entradas do usuário. Jogos como Dwarf Fortress, Minecraft, No Man's Sky (Figura 1.1) e Spore são notáveis por utilizar a geração procedural em partes do seu conteúdo.

Figura 1.1: Captura de tela do jogo No Man's Sky onde planetas inteiros são gerados proceduralmente. Fonte: Hello Games (2016)

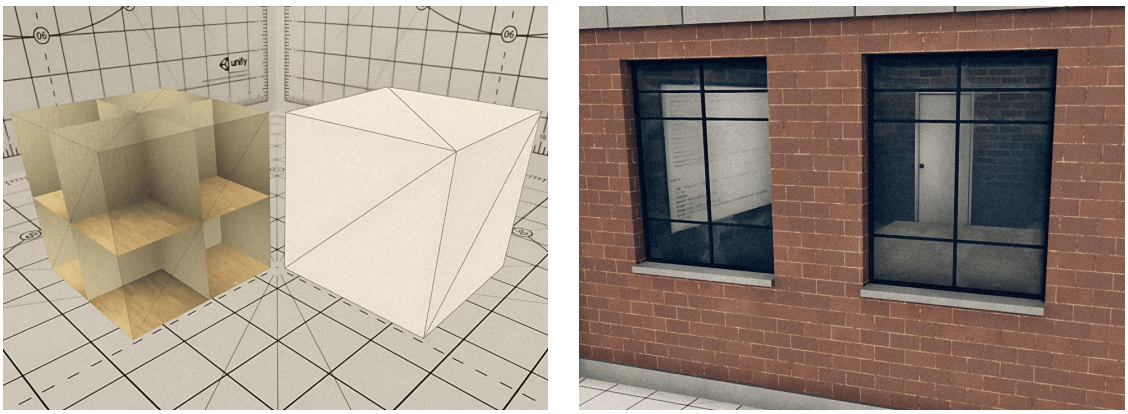
A criação de ambientes digitais requer a colaboração de profissionais de diversas áreas. Artistas desempenham um papel crucial na concepção de objetos que representam elementos nos jogos como casas, árvores, carros e animais. Os *level designers* são responsáveis por incorporar esses elementos ao mundo virtual, enquanto os programadores implementam as mecânicas de jogo que viabilizam a interação com esses elementos. No

entanto, quando a quantidade de conteúdo necessário é muito grande, o projeto pode se tornar inviável devido aos altos custos e tempo de produção, o que pode acontecer tanto em empresas de grande porte com projetos ambiciosos, quanto em empresas menores e desenvolvedores independentes, que enfrentam restrições orçamentárias e acúmulo de funções. Nesse contexto, a Geração Procedural de Conteúdo desempenha um papel significativo, permitindo a criação automática de parte desses elementos. O estudo e o desenvolvimento de técnicas de GPC possibilitam a superação de barreiras técnicas e financeiras, dando mais liberdade criativa aos desenvolvedores.

Uma ambição de desenvolvedores e desejo de muitos jogadores é que mundos digitais como o de jogos das séries Grand Theft Auto (GTA), Assassin's Creed e The Witcher tenham todas as construções acessíveis e seus interiores exploráveis. Grandes títulos de mundo aberto como os citados costumam ter diversas construções acessíveis, em alguns casos a maioria delas, como em The Elder Scrolls Skyrim e no jogo independente Project Zomboid. Entretanto, mapas de jogos como GTA e Assassin's Creed normalmente representam cidades reais com centenas de construções e, mesmo que em escala reduzida, continuam sendo muito grandes e detalhadas. Há vários motivos para que esses produtos não incluam interiores totalmente acessíveis como conteúdo em larga escala. Entretanto, podemos destacar a limitação da capacidade de processamento de consoles e computadores, e a limitação de tempo para criar todo o conteúdo necessário. Alguns jogos modernos utilizam truques com *shaders*, como na Figura 1.2, para criar interiores falsos que são apenas visíveis do exterior e não acessíveis.

Novas tecnologias que permitem o carregamento mais rápido de dados, presentes em consoles como Playstation 5 e o X-Box Series X, em conjunto com tecnologias como o Nanite, presente na Unreal Engine 5, que permite altos níveis de detalhe geométrico em tempo real, podem abrir caminho para que conteúdos como interiores exploráveis com alto nível de detalhe e presentes de forma massiva em mapas sejam viáveis do ponto de vista de processamento. Entretanto, mesmo que possamos exibir esse conteúdo, ainda temos os impedimentos já mencionados ligados ao custo e tempo de produção. Uma forma de abordar esse problema é gerar parte ou todo o conteúdo de forma automatizada.

A geração de construções completas é um tema abrangente e com diferentes fins,



(a) (b)

Figura 1.2: Exemplo de um *shader* para criar a ilusão de interiores reais. Em (a) uma comparação de dois cubos com a mesma geometria mas um deles (esquerda) com o *shader* aplicado, e em (b) uma aplicação. Fonte: Creations (2019)

indo desde a geração do exterior até o posicionamento de móveis e objetos. Dada essa abrangência, este trabalho foca na criação de plantas baixas com ênfase na aplicação em jogos digitais. A limitação ao escopo de jogos é sobretudo para filtrar métodos que abordam este tema do ponto de vista da Arquitetura e Engenharia, onde comumente o resultado desejado deve se aproximar do ótimo, servindo como ponto de partida ou otimização de projetos de estruturas reais. Esse nível de precisão não é necessariamente o que buscamos para os jogos, nos quais o conteúdo gerado deve ser em geral convincente, e não uma réplica exata do real ou uma simulação perfeita. Aspectos como a velocidade de geração e diversidade de resultados, incluído resultados "imperfeitos", podem ser mais desejáveis que tal precisão.

Mesmo limitando-se ao contexto de jogos, é possível encontrar uma boa variedade de publicações e propostas de métodos para geração de plantas baixas. Assim, apesar de haver ferramentas para geração de interiores disponíveis para compra, em alguns casos os desenvolvedores optam pelo desenvolvimento de uma ferramenta própria. Ao optar por essa abordagem, o desenvolvedor irá se deparar com diversos algoritmos, com múltiplas capacidades e limitações. Assim, um estudo dos algoritmos encontrados na literatura e a escolha de um que se adéque aos critérios para implementação em um jogo é relevante na medida em que contribui para um maior embasamento na escolha de métodos que atendam ao propósito mencionado.

1.1 Objetivos

Este trabalho tem como objetivo principal identificar um método para geração procedural de plantas baixas aplicável a um cenário de desenvolvimento de jogos e que atenda a requisitos como facilidade de implementação, possibilidades de melhoria, expansão e integração. Além disso, espera-se que o método atenda a critérios como os definidos por Shaker et al (2016): velocidade, confiabilidade, controlabilidade, expressividade e diversidade, criatividade e credibilidade.

Como objetivos específicos, este trabalho visa: i) estudar e compreender o estado da arte dos métodos de geração procedural de plantas baixas no contexto de jogos; ii) implementar uma versão adaptada da técnica identificada como mais promissora e analisar os resultados obtidos por meio de um conjunto de casos de teste; iii) e desenvolver uma ferramenta de fácil utilização, integrada ao motor de jogos Unity, para geração de plantas baixas a partir de parâmetros definidos pelo usuário.

1.2 Organização do trabalho

O Capítulo 2 deste trabalho apresenta o referencial teórico necessário para a compreensão do tema, abordando conceitos de geração procedural de conteúdo, geração de construções e, mais especificamente, a geração de plantas baixas. Esse capítulo também apresenta uma breve revisão bibliográfica da literatura envolvendo a geração de plantas baixas em jogos, bem como a justificativa para a escolha do método implementado. O Capítulo 3 detalha o processo de desenvolvimento deste trabalho, descrevendo a metodologia adotada, os detalhes da implementação do método, e a geração dos resultados. O Capítulo 4 descreve os casos de teste utilizados e discute os resultados obtidos. Finalmente, o Capítulo 5 apresenta as considerações finais deste trabalho.

2 Referencial teórico

Neste capítulo é feita uma discussão sobre a definição de conteúdo em jogos e sobre a geração procedural em geral. O capítulo também traz uma avaliação breve a respeito das publicações na área e uma revisão da bibliografia focada em geração de plantas baixas. O capítulo se encerra abordando as razões para a escolha do algoritmo proposto por Lopes et al (2010).

2.1 Definição de conteúdo em jogos

Compton et al (2013) descreve o termo "conteúdo" em geração procedural como algo nebuloso, pela dificuldade de se definir o que é ou não é considerado conteúdo em um jogo. Existem diferentes formas de se classificar o que é conteúdo em jogos. Uma resposta curta é dada em Guzdial et al (2022), que define como conteúdo qualquer parte de um jogo que pode ser gerada, incluindo níveis, texturas, enredo, mecânicas e até jogos completos. Entretanto, temos trabalhos que trazem uma definição mais detalhada de conteúdo e dos tipos possíveis de serem gerados.

Hendrikx et al (2013) propõe uma definição de conteúdo de forma hierárquica, em que os elementos de níveis superiores são construídos a partir dos elementos de níveis inferiores. Esta hierarquia é representada na Figura 2.1.

Outra classificação possível é proposta por Liu et al (2021), que foi elaborada visando a aplicação de aprendizado profundo em GPC, mas que continua relevante e serve ao propósito de aqui esclarecer o que é conteúdo, mesmo que técnicas que usam aprendizagem de máquina não sejam o foco deste trabalho. Nesse trabalho os tipos de conteúdo são postos de forma não hierárquica, e são divididos em níveis do jogo (espaços por onde os jogadores se movem), textos, modelos de personagens, texturas, músicas e sons.

Por delimitar bem os tipos de conteúdo e sugerir uma categoria específica para o foco desde trabalho (geração de interiores) será utilizada a definição de conteúdo proposta

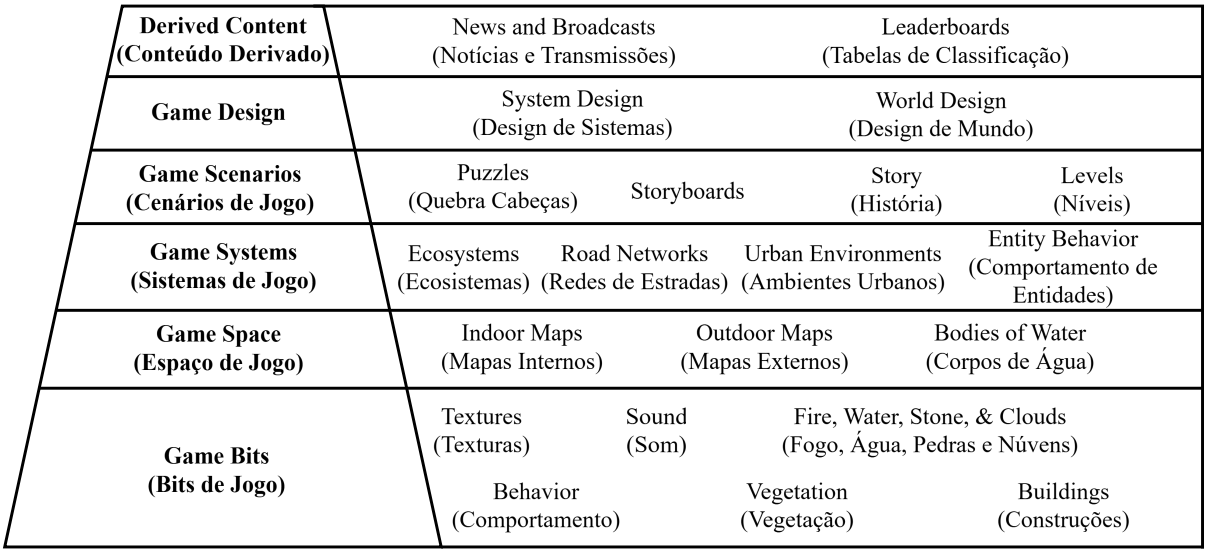


Figura 2.1: Classificação hierárquica de conteúdo. Fonte: Adaptado pelo autor a partir do original em Hendrikx et al (2013).

por Hendrikx et al (2013), conforme ilustrado na Figura 2.1.

2.2 Geração Procedural de Conteúdo

Compton et al (2013) traz um resumo da evolução do termo "geração procedural de conteúdo", ou GPC. Inicialmente, a palavra "procedural" era utilizada para descrever os chamados "*process-derived artifacts*", incluindo o processo de renderização de cenas 3D, como em Rogers et al (1986). Em certo ponto o termo passou a ser adotado pela comunidade de computação gráfica para se referir a abordagens para sintetizar elementos como texturas, modelos e animações. Macri et al (2000) fez com que uma derivação do termo "*Procedural 3D Content Generation*" fosse adotada pela comunidade de desenvolvimento de jogos como "*Procedural Content Generation*".

Atualmente, a Geração Procedural de Conteúdo pode ser definida conforme Togelius et al (2011): "a criação algorítmica de conteúdo para jogos, com entrada do usuário de forma limitada ou indireta." O usuário pode atuar tanto como um designer quanto como o próprio jogador. Adicionalmente, de acordo com Shaker et al (2016), o sistema que gera esse conteúdo deve levar em conta o design, recursos e restrições de jogo, além de o conteúdo precisar ser jogável, ou seja, deve ser possível terminar um nível gerado, subir por uma escada gerada, usar uma arma gerada ou vencer um jogo gerado.

Outro termo relacionado conhecido e que se relaciona com a origem do termo procedural em gráficos é Modelagem Procedural (*Procedural Modeling*), usado, por exemplo, em Müller et al (2006) e por vezes confundido com GPC. O termo é ligado à Computação Gráfica de maneira geral como visto em Ebert et al (2003) e se refere a tipos específicos de conteúdo, como modelos 3D e texturas. Softwares como o SideFX Houdini e Adobe Substance 3D são conhecidos por permitirem criar esse tipo de conteúdo.

Compton et al (2013) descreve o termo GPC como útil para uso na indústria de jogos, mas que pode ser problemático academicamente por excluir certos tipos de artefatos que podem ser gerados. Visando corrigir isso, ele propõe o termo mais geral *Generative Methods*, definido como "uma função que produz artefatos", em que um artefato é algo que contém uma estrutura não trivial e um contexto onde é usado. Com esse termo, busca-se abranger conteúdos não necessariamente ligados a jogos, como *generative art*, *generative music*, *generative architecture*, *generative programming*, e assim por diante.

Como o foco deste trabalho é em geração de plantas baixas para construções em jogos, o termo que será adotado é Geração Procedural de Conteúdo, para que se evite expandir excessivamente o tópico abordado, que está intimamente ligado à arquitetura. Sendo assim, a preferência foi por publicações que adotam essa nomenclatura, com exceção de algumas que, mesmo não focadas em jogos, são relevantes.

2.3 Breve histórico de publicações em GPC

Segundo Togelius et al (2011), o estudo acadêmico de geração procedural é recente em relação ao seu uso pela indústria de jogos, que vem desde a década de 1980. Apenas em 2010 foi realizada a primeira edição do Workshop on Procedural Content Generation (WPCG), evento anual onde são apresentados trabalhos sobre GPC. Também segundo Togelius et al (2011), até a publicação de seu artigo em 2011 não haviam livros-texto ou taxonomias de abordagens básicas para geração de conteúdo em jogos. Em 2016, o livro de Shaker et al (2016) foi publicado, trazendo as abordagens mais comuns em GPC, e em 2022, com cada vez mais técnicas de inteligência artificial sendo aplicadas em geração procedural, Guzdial et al (2022) foi publicado, trazendo abordagens para geração de conteúdo usando aprendizado de máquina.

No que se refere ao tipo de conteúdo gerado, a geração de mapas e níveis é a mais popular, segundo Liapis et al (2020) e Liu et al (2021). Vale ressaltar que essas publicações representam recortes. Liapis et al (2020) se limita a publicações feitas no WPCG, enquanto Liu et al (2021) foca em publicações que fazem uso de Deep Learning. As Figuras 2.2 e 2.3 apresentam os gráficos apresentados pelos referidos autores com o volume de publicações por tipo de conteúdo gerado.

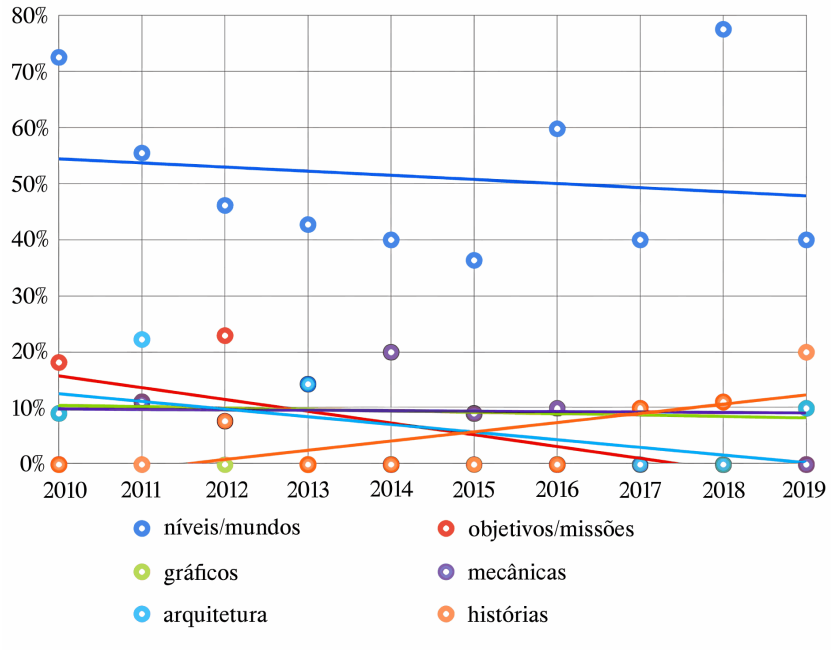


Figura 2.2: Gráfico mostrando o percentual de publicações por ano no WPCG de acordo com seu tipo de conteúdo. Fonte: Liapis et al (2020)

Em relação aos métodos usados para geração, Liapis et al (2020) mostra uma crescente no uso de aprendizado de máquina, o que pode ser verificado na Figura 2.4. Essa crescente se reforça com a publicação de Guzdial et al (2022) como um dos poucos livros sobre Geração Procedural. Apesar da popularidade, por uma questão de escopo este trabalho não se aprofunda em trabalhos que fazem uso dessa família de algoritmos, focando no que Guzdial et al (2022) chama de Geração Procedural de Conteúdo clássica.

Já no campo específico da geração de construções, Kutzias et al (2023) traz um levantamento das publicações sobre o tópico. Embora o seu trabalho não abarque toda a literatura existente na área, o autor destaca o baixo volume de publicações até o fim dos anos 2000. Dos 71 artigos analisados no trabalho, o mais antigo data de 2006, e cerca de 68% deles foram publicados entre 2012 e 2022.

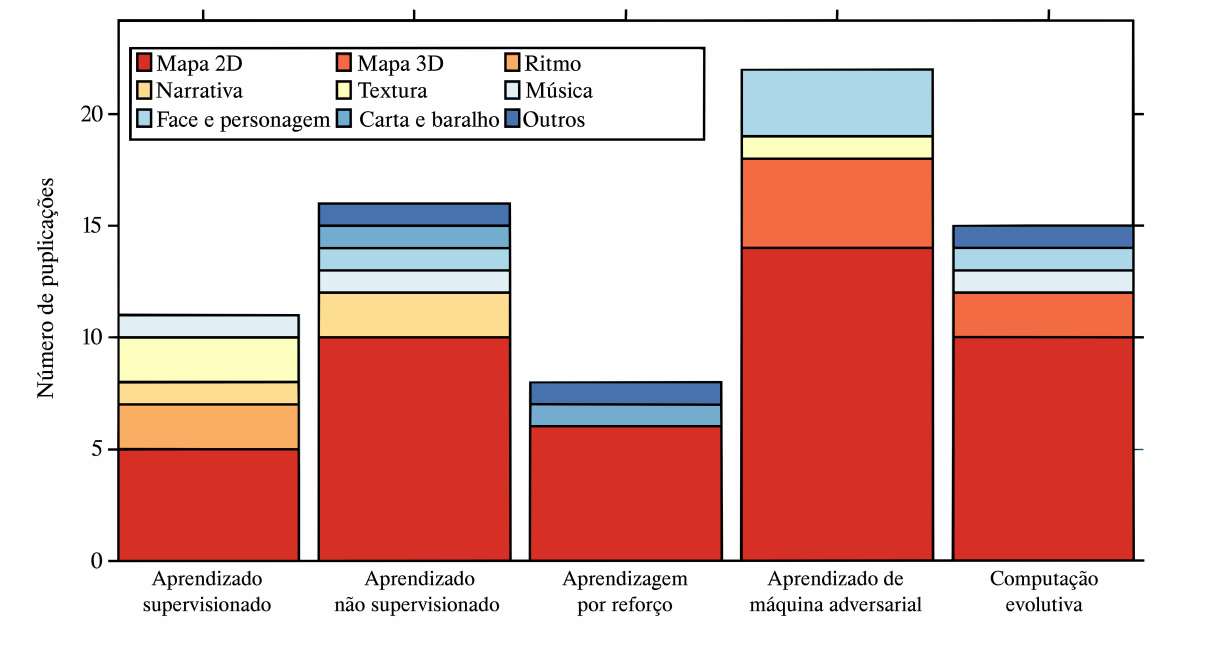


Figura 2.3: Gráfico categorizando os trabalhos analisados de acordo com o tipo de conteúdo e método utilizados. Fonte: Liu et al (2021)

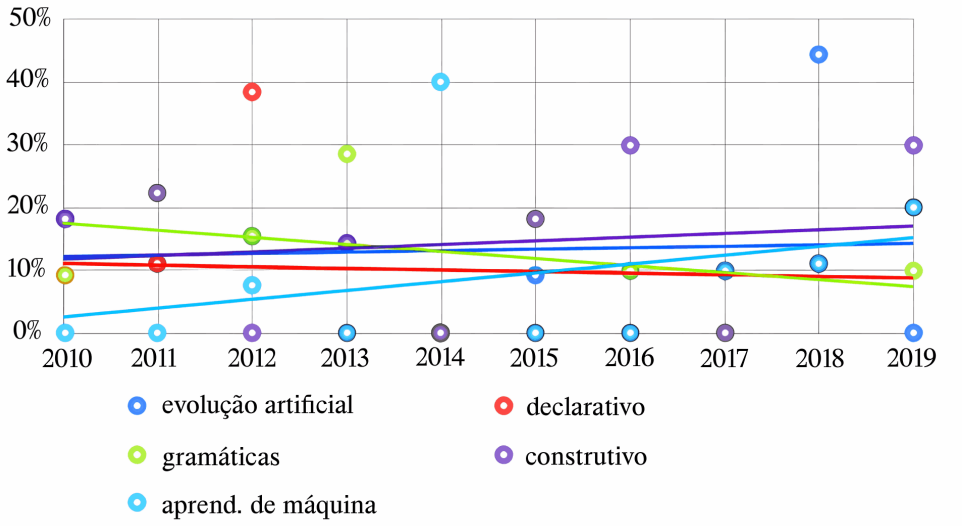


Figura 2.4: Gráfico mostrando o percentual de publicações por ano no WPCG de acordo com as principais famílias de algoritmos dos trabalhos. Fonte: Liapis et al (2020).

2.4 Geração de construções

Um dos elementos possíveis de serem gerados em um mapa virtual são as construções. Diferentes partes desses elementos podem ser criadas de forma procedural, desde apenas a sua fachada até prédios completos com cômodos e mobiliário. Algoritmos de geração de construções não estão restritos à indústria de jogos, tendo também aplicação em áreas

como cinema, engenharia, robótica, e arquitetura.

Diferentes métodos são utilizados dependendo de qual parte da construção é o foco. Algoritmos para geração de exteriores, por exemplo, frequentemente usam gramáticas para geração as formas, como no método proposto por Müller et al (2006). Para geração dos interiores, mais especificamente os cômodos, há métodos que usam estratégias de expansão ou crescimento, como o proposto por Lopes et al (2010), ou gramáticas, como em Leblanc et al (2011). Já com relação ao posicionamento de mobília, Merrell et al (2011) propõe um sistema interativo baseado em uma função de densidade que mapeia um conjunto de diretrizes de design de interiores, com o propósito de sugerir disposição de móveis ao usuário. Também nesse contexto, Germer et al (2009) usa uma abordagem baseada em agentes para mobiliar um ambiente tridimensional em tempo real, à medida que o usuário navega pela cena. É importante ter em mente que essas técnicas não são as únicas e nem exclusivas de cada parte da construção, podendo haver algoritmos aplicados na geração de diferentes partes de uma construção empregando conceitos base similares. Com diferentes técnicas sendo utilizadas, uni-las pode ser um desafio. Trabalhos como o de Tutenel et al (2011) focam justamente na integração de diferentes métodos para criar construções completas.

2.5 Geração de plantas baixas

Segundo Kutzias et al (2023) a geração de plantas baixas consiste na geração, arranjo e conexão de cômodos. A partir de informações iniciais como quais cômodos devem ser gerados, tamanho esperado e conexões (por exemplo, a sala deve estar ligada à cozinha e ao quarto), tem-se, como resultado de algoritmos para este fim, o posicionamento, a dimensão e as conexões dos cômodos.

Ao gerar essas plantas, espera-se que o algoritmo satisfaça certas condições para que o resultado seja válido. Uma das condições mais comuns, usada em trabalhos como o de Lopes et al (2010), é que a conectividade entre os cômodos seja satisfeita, isto é, se foi definido que o cômodo A deve estar ligado ao cômodo B, isto deve estar presente no resultado final. Também é esperado que o algoritmo impeça que existam cômodos desconectados dos demais (como quartos sem portas de entrada, por exemplo). Outras

condições, como tamanho mínimo e formato, também podem ser adicionadas.

2.5.1 Revisão bibliográfica

Nesta seção, são discutidos trabalhos que abordam a geração de interiores, em especial de plantas baixas, para aplicação mais geral, incluso jogos digitais. As técnicas utilizadas nesses trabalhos são variadas, abrangendo aprendizado de máquina, algoritmos de subdivisão espacial e crescimento de regiões.

Por não fazerem parte do escopo deste trabalho, trabalhos que usam técnicas vinculadas à área de Inteligência Artificial são apenas citados de forma breve. Utilizando algoritmos evolutivos, pode-se destacar os trabalhos de Flack et al (2011) e Guo et al (2017), ambos capazes de criar interiores de construções com múltiplos pisos. Já Merrell et al (2010), Guerrero et al (2015), Wu et al (2019) e He et al (2022) usam técnicas de aprendizado de máquina, sendo que o trabalho de Guerrero et al (2015) dá menos ênfase ao interior das construções, apesar de também apresentar resultados relacionados.

Alguns trabalhos utilizam a lógica de subdivisão com a aplicação de gramáticas. O trabalho de Leblanc et al (2011) apresenta uma linguagem para descrever a subdivisão do espaço interior da construção, o que dá muito controle sobre a divisão, mas requer comandos explícitos na linguagem proposta para fazê-la. Marson et al (2010) propõem um método para geração de plantas baixas com o uso de *treemaps* para fazer a subdivisão a partir de uma área inicial retangular, cujos resultados para o método podem ser vistos na Figura 2.5. Mirahmadi et al (2012) apresentam um método similar, inspirado no proposto por Marson et al (2010). Também utilizando *treemaps*, Adão et al (2019) trazem um método capaz de aplicar essa estrutura em construções de área não retangular pré-definidas, além de ser um exemplo da aplicação da geração de construções em projetos ligados ao patrimônio cultural. Outros trabalhos, como os de Willis et al (2021), Zmugg et al (2014), Thaller et al (2013) e Hohmann et al (2010), também propõem métodos que são capazes de gerar os interiores das construções, mas o foco desses é na criação da geometria das construções e na sua representação completa.

Trabalhos como Rodrigues et al (2008) e Sanchez et al (2010) fazem o posicionamento de cômodos a partir de suas regras de conexão e tamanho desejado sem se restringir

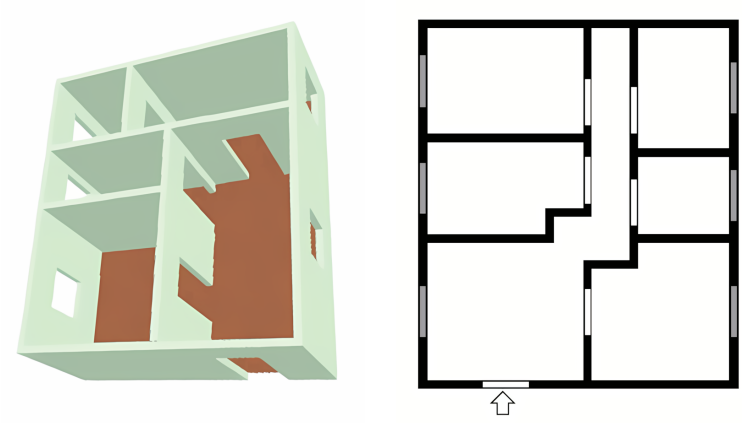


Figura 2.5: Visualização dos resultados de Marson et al (2010) de forma tridimensional e esquemática. Fonte: Marson et al (2010)

a um contorno externo. Ainda com a ideia de posicionar partes da construção a partir de regras de conexão, há o trabalho de van Aanholt et al (2020), que usa o conceito de *Tile Solving* para escolher e posicionar módulos com funções e características arquitetônicas especificas.

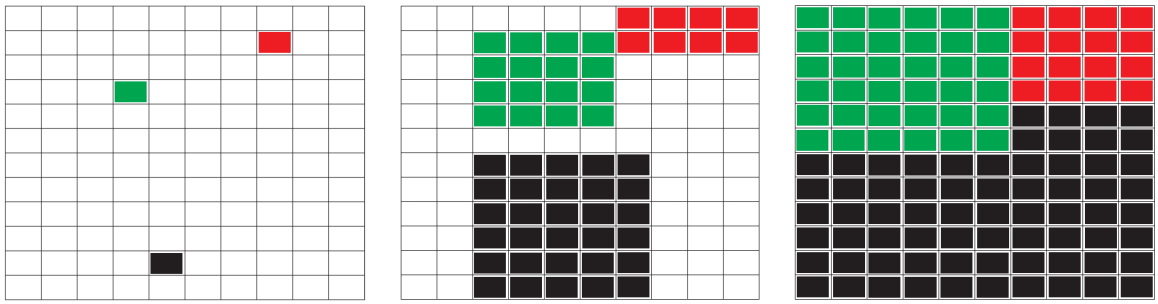
Por último, temos os métodos que utilizam a lógica de crescimento (*growth-based*). Em Tutenel et al (2009), é utilizada essa lógica para expansão de cômodos em uma área predefinida, mas o foco da publicação é em arranjos de mobília. Lopes et al (2010) apresentam um método de geração de plantas baixas por crescimento de cômodos com suporte a contornos externos arbitrários, formas de cômodos irregulares e múltiplos pisos. O algoritmo usa uma grade como base e, a partir de regras de adjacência e conectividade, faz o posicionamento inicial dos cômodos, que em seguida são expandidos até preencher todo o espaço. A proposta de Camozzato et al (2015) funciona de forma similar à de Lopes et al (2010), mas adicionando otimizações para o posicionamento e expansão dos cômodos. Também usando uma lógica similar à de Lopes et al (2010), o trabalho de Freiknecht et al (2019), além de adicionar melhorias à geração do desenho dos cômodos e corredores, faz o posicionamento de escadas e a geração completa da geometria da construção.

2.5.2 Escolha do algoritmo

Após a pesquisa para identificar o estado da arte dos algoritmos para gerar plantas baixas, o algoritmo proposto por Lopes et al (2010) chamou atenção por características como simplicidade, velocidade e flexibilidade, além de ser a base para outros trabalhos que

adicionaram melhorias à solução. Assim, optou-se por explorar o trabalho base, com o propósito de possibilitar futuramente a inclusão das melhorias propostas nos trabalhos subsequentes.

Lopes et al (2010) propõem um algoritmo baseado em regras para gerar os resultados sobre uma estrutura de grade, sendo capaz de chegar à solução final em milissegundos, fazendo com que seja prático em aplicações para geração em tempo real. O algoritmo base é relativamente simples, seguindo uma lógica de expansão dos cômodos. Cada cômodo tem sua primeira célula posicionada próxima aos outros cômodos que devem estar próximos, ou com os quais deveria possuir alguma conexão. Em seguida, são executadas etapas para expandir esses cômodos, primeiro mantendo uma forma retangular e depois em L, até que não haja mais espaço no limite estabelecido, como ilustra a Figura 2.6. Os cômodos são organizados dentro de zonas, conceito que permite agrupar cômodos de setores públicos ou privados de casas.



(a) (b) (c)

Figura 2.6: Imagem com 3 etapas de crescimento para uma área dividida em 3 cômodos (vermelho, verde e preto). (a) mostra a posição inicial dos cômodos, (b) um momento de expansão retangular e (c) o crescimento em L da área em preto. Fonte: Lopes et al (2010)

O algoritmo usa um limite, ou desenho, predefinido do contorno exterior da construção. Isso é ideal para uma ferramenta onde o designer ou artista desenha a forma da construção em um editor de mapas. Essa forma inicial também poderia ser definida por um outro algoritmo.

O uso de uma matriz como estrutura base para posicionamento dos cômodos foi um fator de desempate para a escolha desta abordagem em relação às demais baseadas em subdivisão que possuem capacidades semelhantes de controle e resultado. O uso de uma grade permite a aplicação de pacotes de recursos (*assets*) modulares, muito comuns no desenvolvimento de jogos. *Assets* modulares são objetos que isolados normalmente

não possem função, mas quando combinados formam um grande objeto completo, como pedaços de parede, teto e escadas para formar uma casa, sendo comum serem confeccionados com o mesmo tamanho para facilitar a montagem (por exemplo, pedaços de piso e teto medindo 1m de largura). A Figura 2.7 traz um exemplo de um conteúdo desse tipo.

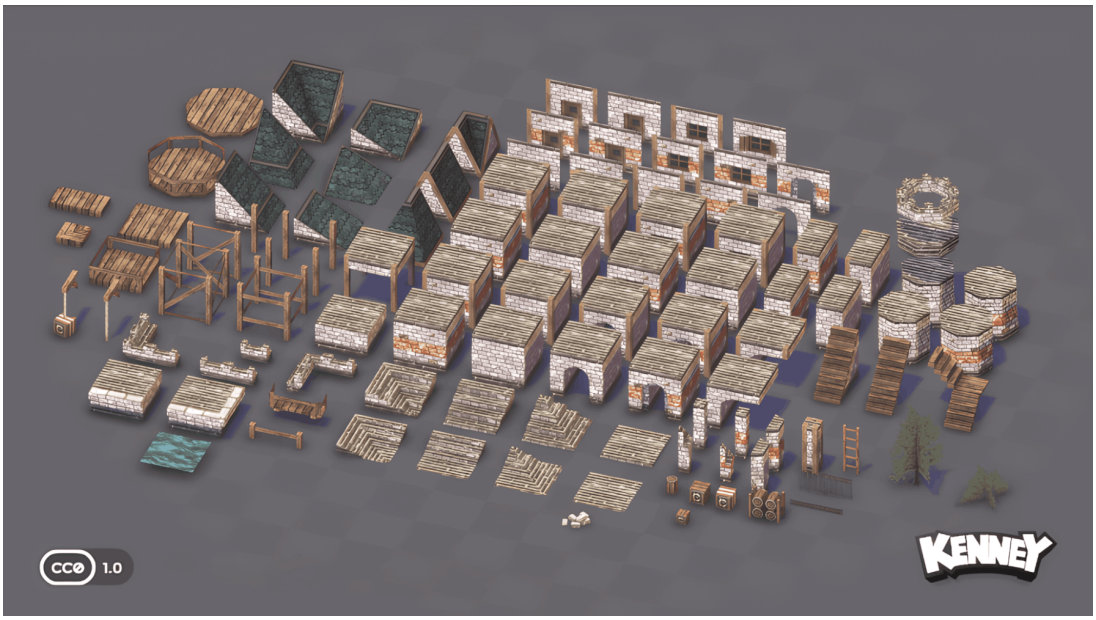


Figura 2.7: Exemplo de um pacote de *assets* modulares criador por Kenney, disponíveis sob a licença Creative Commons CC0. Fonte: Kenney (2025)

Ao usar esses pacotes, é possível, a partir de um mesmo resultado, alternar entre construções com estilo moderno ou renascentista sem mudanças no algoritmo. Muitos trabalhos criam, além da planta baixa, a geometria completa da construção. Isso não só pode limitar a liberdade artística em um projeto se não houver mecanismos elaborados já implementados para modificar a lógica de geração da geometria, mas também estende o tempo de desenvolvimento de uma ferramenta que use a solução. A Figura 2.8 traz um exemplo de resultado obtido por Lopes et al (2010).

O trabalho de Lopes et al (2010) adiciona comportamentos específicos para zonas públicas e privadas e realiza seus testes e avaliação dos resultados focado na representação de casas com arquitetura norte-americana, o que pode não ser adequado para outros contextos. Para resolver isso, durante o desenvolvimento deste trabalho, foram tomadas decisões de implementação com a intenção de generalizar sua aplicação.

(a) (b)

Figura 2.8: Exemplos dos resultados para uma planta complexa no trabalho de Lopes et al (2010). Fonte: Adaptado de Lopes et al (2010)

3 Desenvolvimento

Este capítulo relata o desenvolvimento do software criado neste trabalho para geração de construções. Ele descreve a estrutura geral do sistema, as decisões tomadas durante a implementação do algoritmo de referência e a visualização dos resultados obtidos a partir do que foi desenvolvido.

A partir do algoritmo escolhido foi desenvolvida uma ferramenta[1] para o motor de jogos Unity, que além de ser usada na área foco deste trabalho, fornece ferramentas elaboradas para visualização dos resultados e criação de interfaces.

3.1 Modelagem do sistema

O sistema foi projetado visando manutenção e expansão futura, ao reduzir as dependências entre diferentes partes. A interface de usuário e funcionalidades de visualização podem ser substituídas sem necessidade de alteração no algoritmo principal. Em alto nível, o usuário a partir da interface da Unity gera os dados que serão usados pelo algoritmo, e esses dados em seguida são enviados para o gerador que irá criar novas informações com os resultados. No fim, uma classe de visualização lê esses dados e os exibe na cena. A Figura 3.1 traz um diagrama com essa lógica em mais detalhes, enquanto a Figura 3.2 mostra o editor da Unity com as janelas customizadas desenvolvidas para esta aplicação.

Dentre as principais classes que compõem o sistema, a `FloorPlanManager` é a que armazena a planta baixa de um nível ou piso da construção. Nela, um dicionário armazena as regras de adjacência, onde cada elemento armazena as zonas adjacentes à sua própria zona.

A classe `Grid` armazena as células que compõem a área total da planta, fornecendo um meio de acessá-las em sua posição na matriz. Já a classe `Zona` representa uma subdivisão da área total do piso. Essa classe também armazena informação ligada ao uso daquela área, como por exemplo banheiros, quartos, jardins e outros. As zonas são

[1] Disponível em https://github.com/LeandroDornela/floor-plan-generator

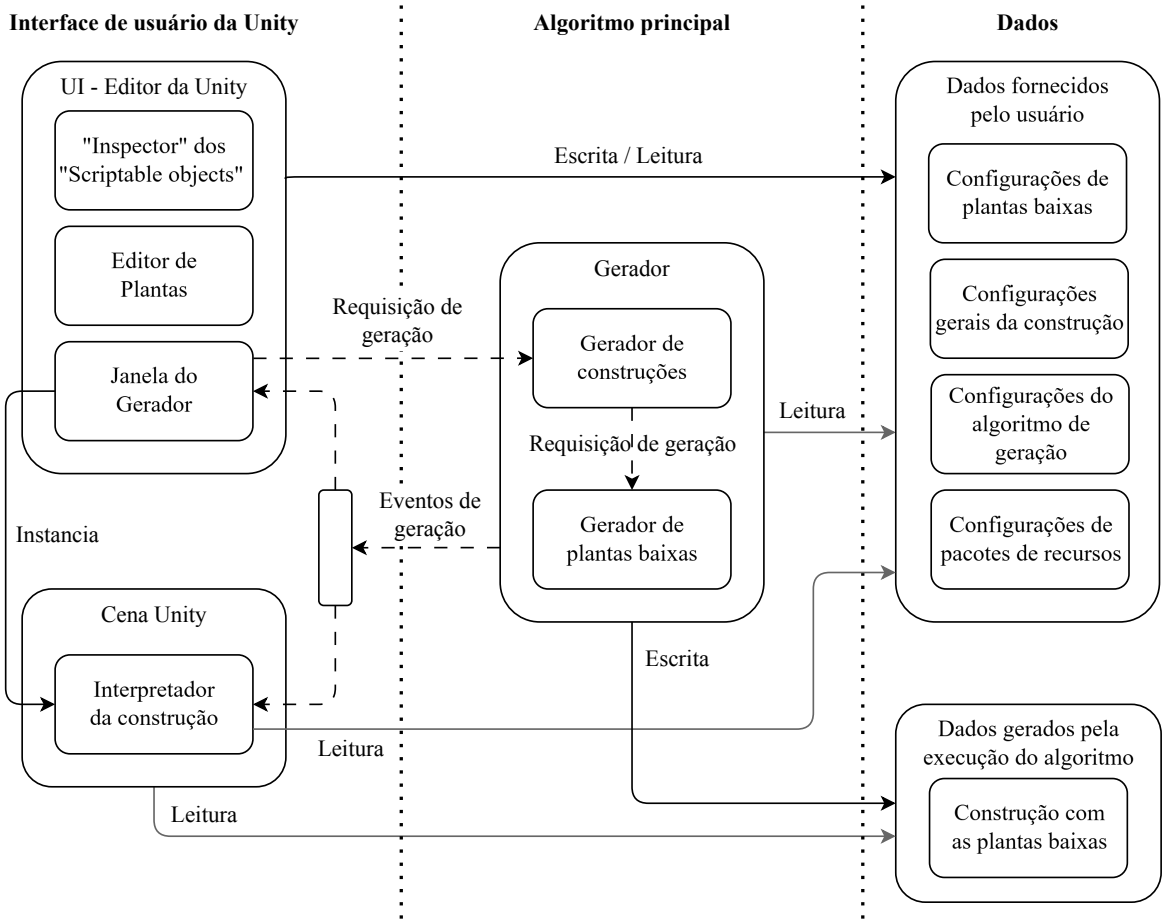


Figura 3.1: Visão geral do sistema. Fonte: Do autor.

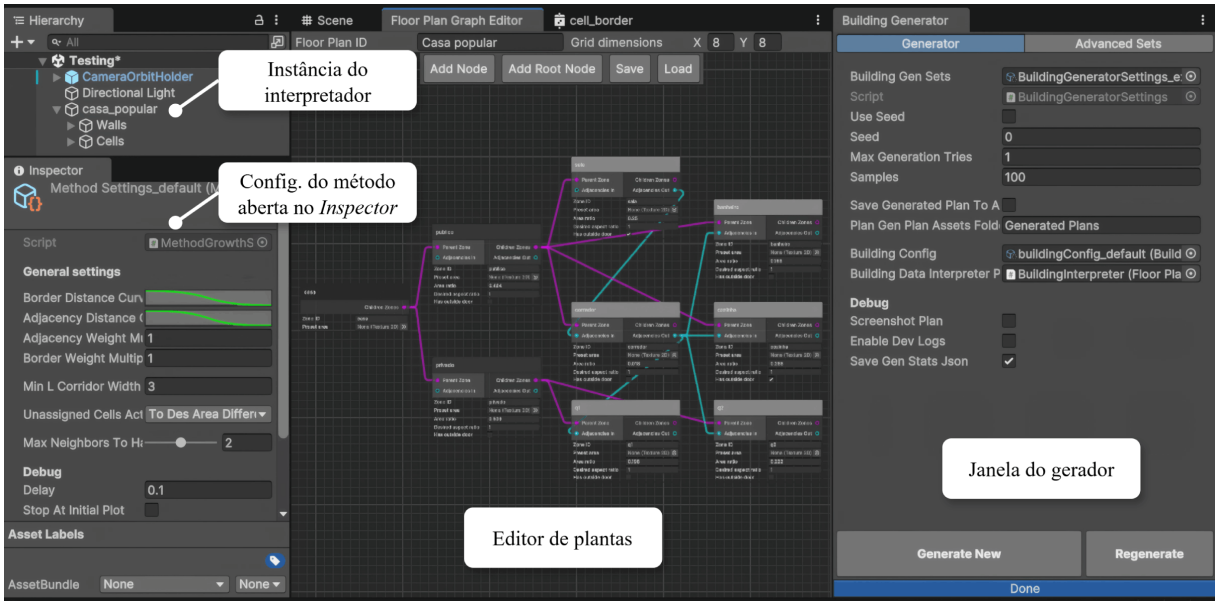


Figura 3.2: Editor da Unity com as ferramentas desenvolvidas em destaque. Fonte: Do autor.

organizadas de maneira hierárquica, de forma que uma zona pode ser subdividida em outras zonas. Na publicação de Lopes et al (2010), o termo "zona" é usado exclusivamente no contexto de setorização da construção, com subdivisão de uma casa em setores como social, íntimo ou de serviço, havendo uma diferenciação entre zonas e cômodos. Neste trabalho, a ideia de zona foi generalizada, com todos os espaços criados sendo zonas, e as folhas da estrutura hierárquica de zonas são os cômodos, não havendo uma classe dedicada a representá-los. Além disso, as zonas armazenam referências para instâncias da classe `Cell` que representam pequenas áreas retangulares, subdivisões do piso da construção.

Por fim, vale mencionar a classe `WallTuples`, que armazena uma dupla de células, cada uma pertencente a uma zona. Um objeto dessa classe representa o limite entre duas zonas, interpretado como uma parede ou porta.

3.2 Execução do algoritmo

Antes de avançar com a execução, alguns termos usados para identificar relacionamentos entre as zonas devem ser definidos. Durante a execução do algoritmo, para cada zona Z é necessário ter referências para outras quatro zonas hierarquicamente relacionadas a Z. Para facilitar o entendimento, foi utilizada a nomenclatura apresentada na Figura 3.3.

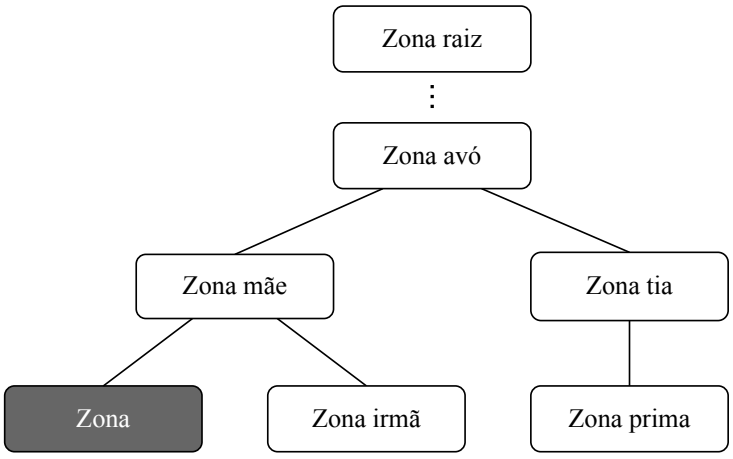


Figura 3.3: Nomenclatura para relações hierárquicas entre as zonas a partir de uma zona de referência em cinza escuro. Fonte: Do autor.

O primeiro passo para gerar uma planta baixa é obter os dados necessários. Entre esses dados estão as configurações gerais do algoritmo, a configuração da planta baixa

do piso e a configuração da representação gráfica da construção. Nas configurações de uma planta baixa se encontram a dimensão da grade de células, uma lista com as zonas adjacentes de cada zona, e uma lista com as zonas a serem geradas. O usuário também especifica a hierarquia dessas zonas. Além disso, para cada zona a ser gerada, também há a sua identificação, a porcentagem da área total que deve ser ocupada por ela, e a sua zona mãe. O usuário também pode informar, para a grade atribuída à zona raiz, texturas opcionais contendo a predefinição da área útil total dessa grade. Texturas também podem ser fornecidas para definir áreas para as primeiras subdivisões da zona raiz. A Figura 3.4 mostra a configuração para uma planta simples que serve como exemplo de execução do algoritmo neste capítulo.

No trabalho de Lopes et al (2010), adjacências entre cômodos dentro de diferentes zonas não podem ser definidas pelo usuário, apenas entre cômodos e zonas mãe. Por exemplo, o corredor de um setor privado deve estar conectado a uma zona pública. Já neste trabalho, a implementação permite a definição de adjacências entre zonas (cômodos) que pertençam a zonas mães diferentes, aqui chamadas de *zonas primas*. Isso dá mais controle ao usuário que, dessa forma, pode definir explicitamente que, por exemplo, a sala que está no setor público da casa deve estar conectada ao corredor que está na seção privada.

A partir dos dados fornecidos se inicia a execução do algoritmo. O primeiro passo é definir a área ocupada pela zona raiz. Se uma textura representando a área inicial foi fornecida, píxeis em branco são interpretados como células que pertencem à zona, e píxeis pretos como células que não pertencem. Se nenhuma imagem foi definida, todas as células da grade serão atribuídas à zona raiz. As células não atribuídas à zona raiz nesta etapa não serão usadas nas etapas posteriores, sendo consideradas área não útil. Na sequência, se inicia o processo de posicionamento inicial e expansão das zonas. A primeira zona a ser subdividida é adicionada a uma fila de zonas a subdividir. Os filhos da primeira zona adicionada a esta fila são adicionados a uma lista de zonas a expandir, e para cada uma destas zonas a expandir serão executadas as etapas de posicionamento inicial, expansão retangular, expansão livre e expansão em forma de L.

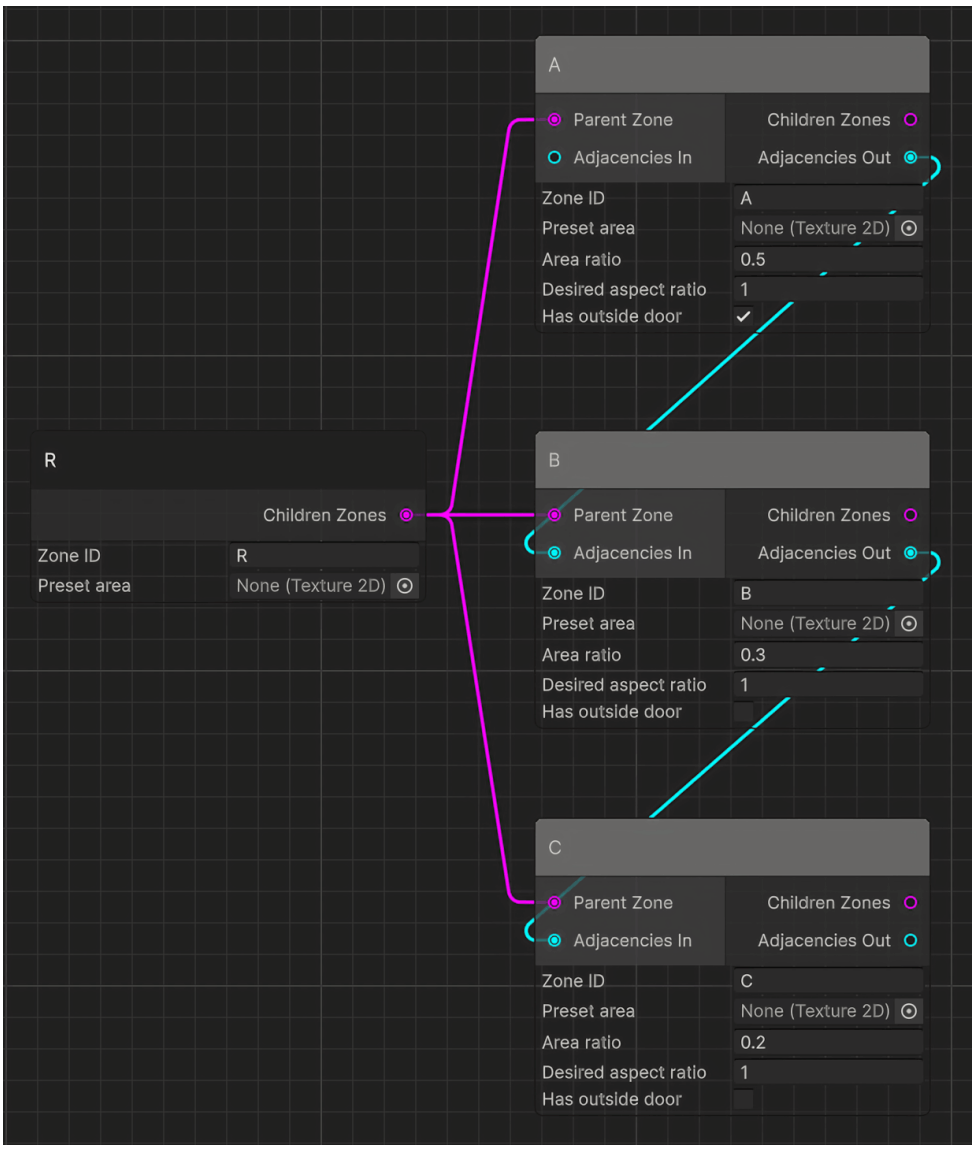


Figura 3.4: Entrada para caso de teste simples, com 3 subdivisões (A, B e C) da zona raiz (R). Conexões em rosa indicam a hierarquia das zonas. Em ciano estão as indicações de adjacências, representadas de forma unidirecional. Neste exemplo, a geração deve incluir conexões entre A e B e entre B e C. Fonte: Do autor.

3.2.1 Posicionamento inicial das zonas

No posicionamento inicial, cada zona da lista da atual iteração terá a posição da sua primeira célula definida usando uma matriz de pesos. Os pesos serão calculados usando as regras de adjacência, o tamanho esperado de cada zona e as bordas da área disponível. Para otimizar o cálculo, apenas as células pertencentes à zona mãe da zona a ser posicionada serão consideradas candidatas. Os valores dos pesos são maiores ou iguais a 0, e quanto maior for o valor maiores são as chances de uma célula na matriz ser escolhida.

A posição inicial deve estar próxima das bordas para aumentar as chances das

zonas terem acesso a áreas externas, mas não próximas demais. No cálculo do peso das bordas, uma margem de segurança inicial é calculada a partir da área desejada da zona para evitar que ela seja posicionada muito próxima. Segundo Lopes et al (2010), isso gera formatos mais plausíveis. Primeiro, obtêm-se um valor desejado estimado para o lado da zona (L_z) a ser posicionada. Como não é possível saber a forma final de uma zona, assume-se idealmente que possuirá lados iguais. Isso facilita o cálculo, além de ser o suficiente para estimar as distâncias necessárias. O valor de L_z é encontrado a partir da Equação 3.1:

$$L_z = \sqrt{r_a \cdot a_t}, \tag{3.1}$$

onde r_a é a porcentagem da área total que deve ser ocupada pela zona e a_t é a área total disponível na grade.

Em seguida, para cada célula na matriz de pesos, é buscada sua distância para a célula de borda mais próxima. Se a distância for menor que um valor mínimo D_{minB} ou maior que D_{maxB}, o peso para células a esta distancia será 0. Os valores de D_{minB} e D_{maxB} são obtidos em função de L_z pelas equações 3.2 e 3.3:

$$D_{minB} = \frac{L_z}{4}, \tag{3.2}$$

$$D_{maxB} = L_z, \tag{3.3}$$

onde o denominador 4 é um valor obtido de forma empírica que deve ser menor que o denominador que será usado para o cálculo de pesos das zonas irmãs na 3.4. A ideia é que a área mínima de segurança das bordas seja menor do que das zonas irmãs, para evitar que zonas adjacentes à que está sendo posicionada sejam colocadas entre ela e a borda, resultando em mais erros na geração. A Figura 3.5 traz uma representação dos pesos nesses diferentes intervalos de distância.

Caso a distância mais curta entre a célula a verificar e a borda esteja entre D_{minB} e D_{maxB}, o valor do peso é calculado usando uma curva definida pelo usuário. A Figura 3.6a mostra a curva utilizada durante o desenvolvimento, definida no editor da Unity.

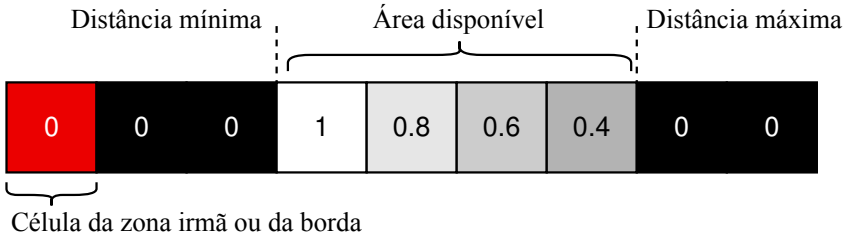
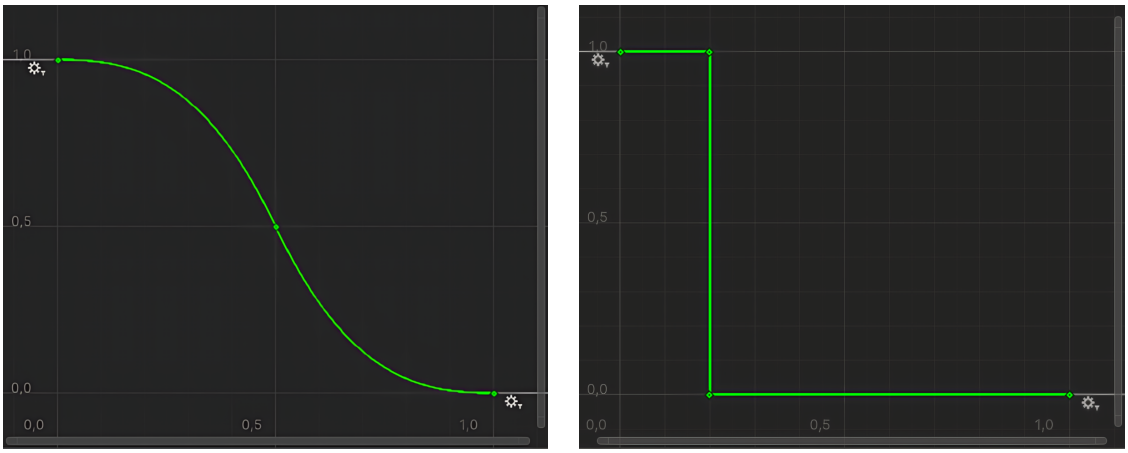


Figura 3.5: Ilustração de uma sequência de células com os respectivos pesos e indicação dos limites usados no cálculo. Em vermelho, a célula de referência; em preto, as células com peso 0 fora da área disponível, seja dentro da distância de segurança mínima, ou além da distância máxima; e em tons de cinza, a área ou intervalo com as células que podem ter peso maior que 0. Fonte: Do autor.

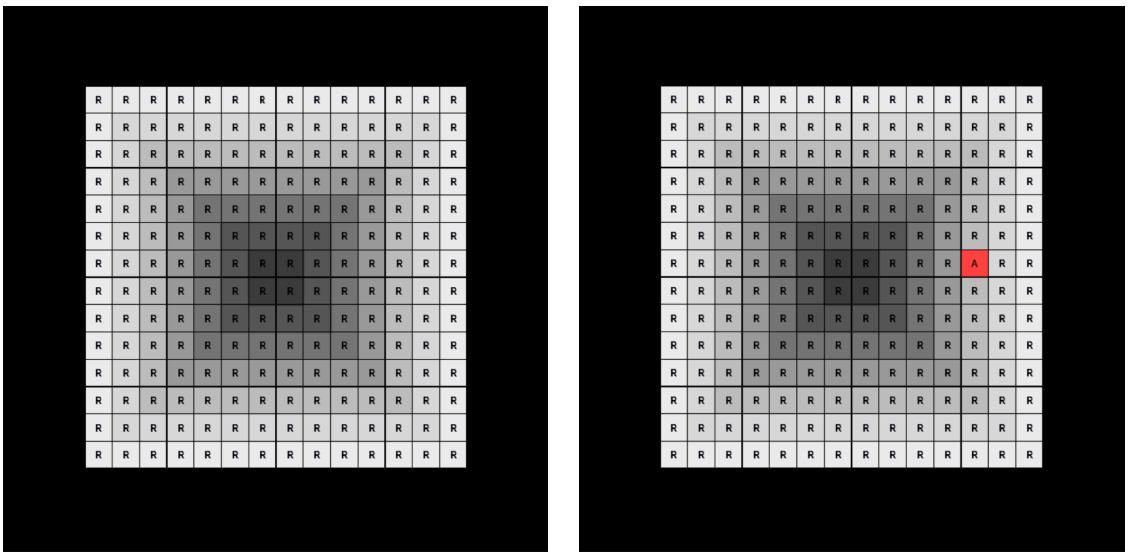
Com essa curva, células mais próximas do limite mínimo da borda tendem a ter maior peso, e esse peso decai rapidamente para valores inferiores conforme se afasta. O usuário pode alterar a curva para um ajuste fino nos pesos. A Figura 3.6b traz um exemplo de curva que resulta no posicionamento inicial da zona em uma faixa estreita de células com o mesmo peso. A Figura 3.7 mostra a grade de pesos do exemplo da Figura 3.4 para posicionamento da primeira zona filha da zona raiz. Como não há outras zonas já posicionadas, apenas o peso das bordas tem influência.



(a) (b)

Figura 3.6: Janela do editor da Unity com exemplos de curvas utilizadas para cálculo do peso das células da borda para cada célula da matriz de pesos. (a) resulta em um decaimento de pesos suaves em direção ao centro da área e (b) resulta em uma faixa estreita próxima à borda. Fonte: Do autor.

A próxima etapa no cálculo de pesos é determinar o efeito das zonas primas ou tias. Essa etapa é o que permite criar adjacências ou conexões entre zonas de diferentes zonas mãe. Aqui, para cada zona que deve ser adjacente à zona que será posicionada,



(a) (b)

Figura 3.7: Em (a) temos a representação da matriz de pesos para posicionamento da zona A. Em (b), em vermelho, temos a célula escolhida para A. Valores mais claros possuem mais chances de serem escolhidos. Fonte: Do autor.

verifica-se se a zona adjacente já passou pelo processo de expansão. Se não passou, considera-se que sua posição ainda é indefinida e será usada a zona tia para cálculo. Dessa forma, quando a então zona adjacente for posicionada, a zona atual será quem influenciará na sua posição inicial. A Figura 3.8 ilustra os pesos para um caso de adjacência entre zonas com mães diferentes. O cálculo segue similarmente ao do cálculo de peso das bordas iniciais, mas agora as células usadas são as das bordas da zona adjacente, seja prima ou tia. Mais uma vez, a curva representada na Figura 3.6 é usada para determinar o peso quando a célula está a uma distância entre D_{minB} e D_{maxB}. O valor obtido é multiplicado pelo valor obtido na etapa anterior e definido como o novo peso para aquela célula.

Por fim, é calculada a influência das zonas irmãs já posicionadas. O cálculo é mais uma vez similar ao das bordas, onde uma distância mínima D_{minI} é calculada mas, desta vez, com base no tamanho estimado do lado da zona irmã já posicionada (L_{zI}). A distância D_{minI} é obtida utilizando a Equação 3.4:

$$D_{minI} = \frac{L_{zI}}{2}, \tag{3.4}$$

onde o denominador 2 deve ser menor que o denominador da Equação 3.2 para D_{minI} ser maior que D_{minB} para evitar o posicionamento da zona entre a irmã e a borda. Também é

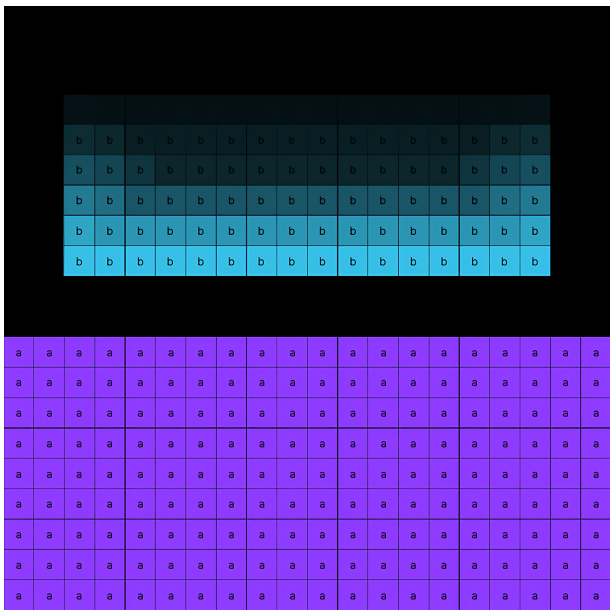


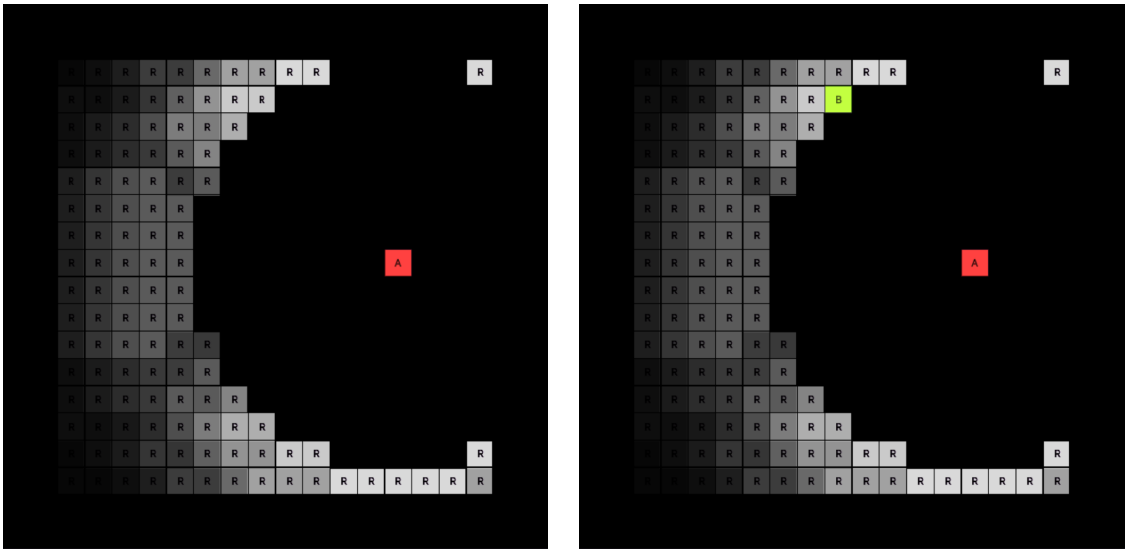
Figura 3.8: Exemplo de pesos (metade superior) para posicionamento de uma zona c filha de b (azul) adjacente a uma zona filha de a (lilás). Nesse momento a não foi subdividida, então o peso para c será com base em a, e não em sua filha. Tons de azul mais claro têm mais chance de serem escolhidos. Fonte: Do autor.

usada uma distância máxima D_{maxI} para garantir que zonas adjacentes fiquem próximas. Essa distância máxima é obtida a partir da Equação 3.5:

$$D_{maxI} = D_{minI} + L_z \tag{3.5}$$

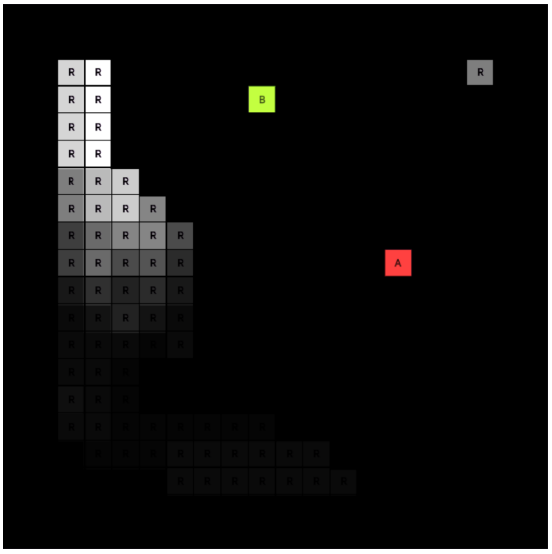
Para o caso onde a zona já posicionada não é adjacente à atual, somente são zerados os pesos das células ao seu redor, a uma distancia obtida também pela Equação 3.4 e não há alteração dos pesos para além desta distância. Já para aquelas que são adjacentes, é usada uma segunda curva para determinar o peso entre a distância mínima D_{minI} e a máxima D_{maxI}. Durante o desenvolvimento foi usada uma curva com características similares à usada no cálculo de bordas representada pela Figura 3.6. Assim como no cálculo de bordas primas, o valor de pesos é multiplicado e atualizado. A Figura 3.9 mostra a matriz de pesos resultante no final desta etapa para as zonas B e C no exemplo utilizado.

Após essas três etapas, obtém-se um peso total para cada célula, representado de forma simplificada pela Equação:

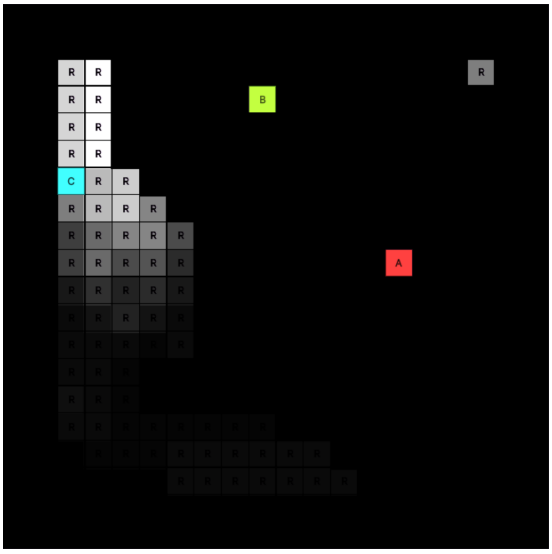


(a)

(b)



(c)



(d)

Figura 3.9: Em (a) a matriz de pesos para a zona B. Note a margem de segurança ao redor de A, bem como pesos maiores ao redor desta margem, já que B foi definida como adjacente a A. Em (b), em verde, a célula escolhida para B. O mesmo se repete para a zona C, mas tendo B como adjacente. Em (c) os pesos para posicionamento de C e em (d), em ciano, a célula escolhida. Fonte: Do autor.

$$P_c = P_b \cdot P_p \cdot P_i, \tag{3.6}$$

onde P_c é o peso final da célula, P_b o peso das bordas da zona mãe, P_p o peso de zonas primas ou tias adjacentes, e P_i o peso de zonas irmãs.

Para os casos raros em que no fim do processo não há nenhuma célula candidata com peso maior que 0, ao contrário de parar o algoritmo, tenta-se salvar a execução escolhendo uma célula qualquer da grade aleatoriamente. Esse tratamento se mostrou necessário nos casos em que grades ou zonas a serem subdivididas eram muito pequenas, pois as distâncias mínimas de segurança acabam por anular qualquer candidato. Havendo células candidatas, é escolhida uma e segue-se para a próxima zona a ser posicionada.

3.2.2 Expansão das zonas

Com a primeira célula de cada zona da lista de zonas a expandir posicionada, inicia-se a etapa de expansão, ou crescimento das zonas. A cada iteração desta etapa, será escolhida uma zona para expandir. Aqui mais uma vez se usa a ideia de pesos para a escolha, sendo que zonas com maior área desejada possuem pesos maiores. Assim, zonas que devem ocupar mais espaço serão chamadas mais vezes para expansão. Em um primeiro momento, a expansão se dá de forma a manter a zona com um formato retangular, respeitando a sua razão de aspecto, por padrão 1:1 ou simplesmente 1. Ao atingir a área desejada ou não havendo mais espaço, a zona para de crescer e a execução segue para as outras zonas até que nenhuma possa se expandir. O Algoritmo 1 mostra com mais detalhes o processo de expansão das zonas.

Algoritmo 1: Algoritmo principal com destaque ao processo de expansão.

Fonte: Adaptado de Lopes et al (2010).

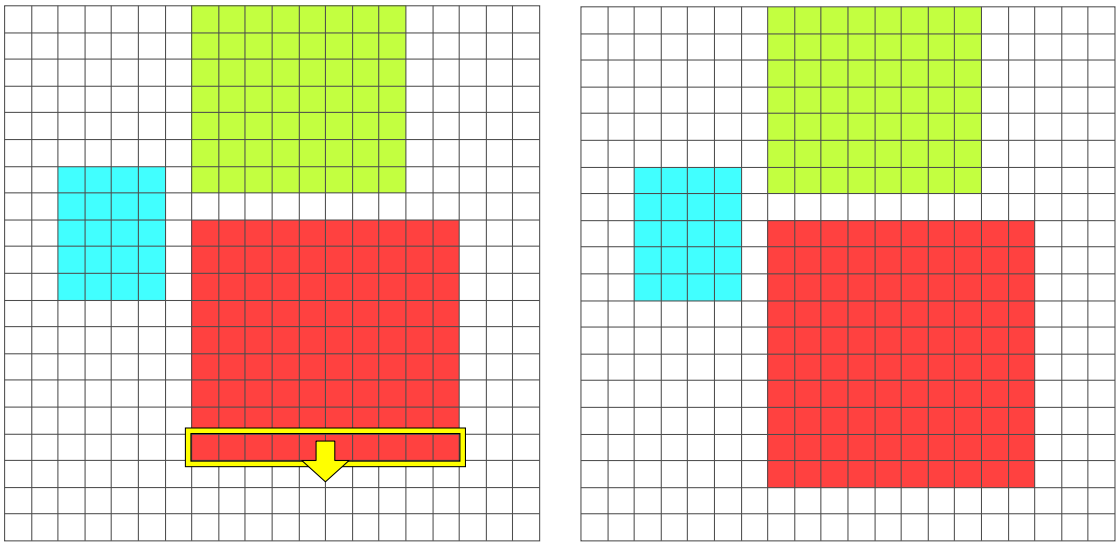
```
Data: Zona raiz r
Data: Grade de células m
Result: Planta baixa definida em m
início
    AtribuirCelulasRaiz(r, m);
    zonasASubdividir ∪ {r};
    enquanto zonasASubdividir ≠ ∅ faça
        zonaSendoDividida ←
        ObterZonaASubdividir(zonasASubdividir);
        zonasASubdividir \ {zonaSendoDividida};
        zonasAExpandir ← ObterZonasAExpandir(zonaSendoDividida);
        PosicionarZonas(zonasAExpandir);
        enquanto zonasAExpandir ≠ ∅ faça
            zona ← EscolherZona(zonasAExpandir);
            se ¬ ExpansãoRetangular(zona, m) então
                zonasAExpandir \ {zona};
            fim se
        fim enqto
        zonasAExpandir ← ObterZonasAExpandir(zonaSendoDividida);
        enquanto zonasAExpandir ≠ ∅ faça
            zona ← EscolherZona(zonasAExpandir);
            se ¬ ExpansãoL(zona, m) então
                zonasAExpandir \ {zona};
            fim se
        fim enqto
        zonasASubdividir ∪ ObterZonasAExpandir(zonaSendoDividida);
    fim enqto
    PreencherEspaçosVazios(m);
    TesteConectividade(m);
fim
```

Durante a expansão, o lado escolhido para expansão é aquele que resultará em uma razão de aspecto mais próxima do desejado. Em todas as situações, sempre há no mínimo 2 lados para onde uma zona pode se expandir. Como critério de desempate, expande-se para o lado onde há mais espaço potencial para expansão. Esse espaço é aquele que possui o maior número de linhas de mesmo comprimento paralelas ao lado testado, seguindo na direção perpendicular ao lado. Havendo empate nesse critério, é escolhido um lado aleatório. A expansão na direção com mais espaço potencial busca gerar melhores resultados, evitando que zonas cresçam usando o espaço de zonas vizinhas, o que poderia restringir o crescimento dessas.

Para encontrar uma fileira de células válida para expansão, percorre-se cada célula da lateral testada verificando a célula vizinha na direção de expansão desejada. Se a célula pertence à mãe da zona a ser expandida ela é considerada válida. Como a célula só possui referência a apenas uma zona, pertencer à zona mãe implica em não estar em uma zona irmã, além de evitar que células de zonas tias ou da área não útil sejam consideradas como candidatas, mantendo o processo de expansão confinado na zona mãe. Caso seja encontrada uma fileira completa de células livres, repete-se o processo até que não as encontre mais. O número de fileiras completas indica o espaço potencial de expansão da zona a partir do lado testado. Para um lado ser considerado válido e apto à expansão nesta etapa, deve haver no mínimo uma fileira completa de células vizinhas disponível.

A Figura 3.10 mostra a expansão de uma fileira de células da zona A, em vermelho. Neste momento da execução, a zona A possui largura maior do que a altura, então para manter a proporção de 1:1 a expansão deve ser vertical. O maior espaço de expansão está na parte inferior da zona, portanto a fileira inferior é a escolhida para expansão, como destacado na Figura 3.10a.

Após todas as zonas atingirem o tamanho mínimo desejado ou não possuírem mais espaço para expandir mantendo a proporção definida, inicia-se a expansão retangular livre. Nessa etapa, expande-se similarmente à anterior, buscando pelo maior espaço potencial, ignorando o aspecto da zona. O objetivo é começar a preencher o espaço ainda livre. Quando não é mais possível se expandir de forma retangular, cada zona tenta expandir com formato de L. Para definir se essa expansão é possível, é feita a busca em cada lado



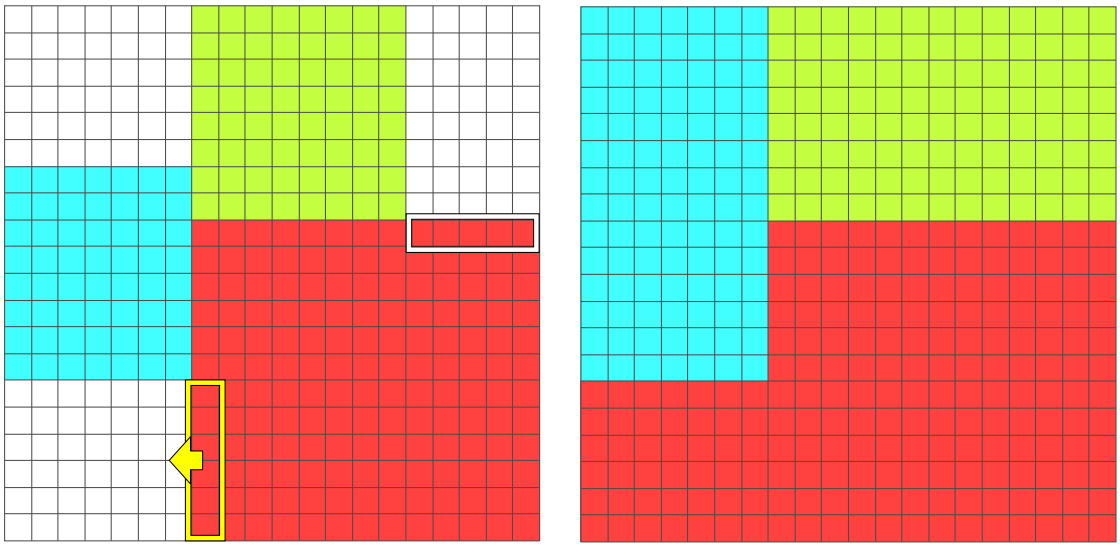
(a) (b)

Figura 3.10: Expansão da fileira de células inferiores da zona vermelha A para a região com mais espaço potencial de expansão e respeitando o aspecto desejado da zona. Em (a) a linha selecionada e (b) o resultado da expansão. Fonte: Do autor.

da zona pela maior cadeia de células vizinhas livres que comece na primeira ou ultima célula da lateral e tenha um número mínimo de células (definido pelo usuário). O objetivo é evitar uma expansão em L que crie corredores estreitos. Encontrada essa sequência, a zona é marcada como tendo forma de L e segue expandindo apenas a cadeia de células que deu início a essa forma. Quando não há mais espaço para continuar esta expansão em todas as zonas, a etapa de crescimento termina.

O inicio da expansão em L é ilustrado na Figura 3.11. Na Figura 3.11a, tem-se 2 fileiras aptas a se tornar a "perna do L". Como a inferior esquerda possui mais células, esta é a escolhida, e a zona A segue sua expansão somente nessa fileira. A Figura 3.11b mostra o estado final da expansão livre para as zonas B e C e da expansão em L para a zona A.

As diferentes etapas de crescimento não garantem que todas as células disponíveis da grade tenham sido atribuídas a uma zona folha. Para resolver isso, a grade é varrida em busca de grupos contínuos de células livres. Encontrado um grupo, são buscadas todas as zonas adjacentes a esse grupo, e uma delas é escolhida para receber todas as células. O critério de escolha é definido pelo usuário, podendo ser a zona com maior distância da área desejada, a zona com o maior número de células adjacentes ou o descarte das células.



(a) (b)

Figura 3.11: (a) Expansão em L para a zona A, começando pela área destacada em amarelo identificada por uma seta, e a expansão livre retangular para as zonas B e C. (b) Resultado final da expansão. Fonte: Do autor.

3.2.3 Teste de conectividade

Com todas as células válidas da grade atribuídas a uma zona folha, é preciso verificar a conectividade entre os cômodos. Nesse momento, na implementação de Lopes et al (2010), os autores deixam de usar a estrutura de grade e passam a trabalhar com as paredes, sem esclarecer qual a estrutura de dados usada. Aqui, foi mantida a estrutura de grade, já que não se observou um impedimento para a verificação de adjacências e conectividade.

Nesta implementação, as regras de adjacência e conectividade foram unificadas. Toda zona que deve ser adjacente a outra também deve estar conectada por uma porta. Na prática, esta abordagem gerou resultados similares aos do trabalho de Lopes et al (2010). Caso necessário, pode ser facilmente adicionada uma nova variável à regra, já que isso não altera a execução anterior do algoritmo, definindo apenas se deve ou não ser adicionada uma porta entre zonas adjacentes na etapa corrente.

As regras de adjacência, então, são usadas para definir quais zonas folhas estão conectadas entre si. Para cada zona, evitando verificações redundantes, varre-se suas células de borda verificando os vizinhos. Caso a célula pertença a um vizinho e ele esteja entre os que devem estar conectados à zona, verifica-se se a célula da zona ou do vizinho são aptas a possuir uma porta. Em caso positivo, uma tupla candidata a possuir a porta é criada com ambas as células. Nesse momento, também é verificado se a zona deve ter

uma porta para o exterior. Se sim, a área exterior de células não úteis é tratada como uma zona para que seja criada uma tupla válida. Para uma célula ser candidata a porta, ela deve ter até 2 vizinhos pertencentes a sua zona. Essa condição permite posicionar portas próximas a cantos.

Aqui vale destacar outra diferença da implementação de Lopes et al (2010), onde são feitas conexões que não foram explicitamente declaradas pelo usuário. Um exemplo é quando cômodos em uma zona pública não conectados a nenhum outro cômodo são conectados automaticamente a qualquer outro cômodo possível na zona pública. Para generalizar o conceito das zonas e evitar conexões indesejadas, optou-se por apenas realizar as conexões definidas pelo usuário.

Com todas as tuplas de candidatos a porta definidas, são escolhidas aleatoriamente uma tupla para cada par de regras de adjacência. Se todas as zonas adjacentes tiverem ao menos 1 candidato a porta, a conectividade entre as zonas é satisfeita. Ressalta-se que o algoritmo utiliza as regras definidas pelo usuário. Assim, caso uma zona folha (cômodo) não possua uma regra de adjacência, ele ficará inacessível. Uma validação dos dados de entrada seria necessária para evitar este problema. Ainda nesta etapa, junto da verificação das candidatas a porta, quando a célula pertence a outra zona qualquer e não é uma candidata a porta, a tupla de células é obrigatoriamente a representação de uma parede.

Ao final desta etapa, obtém-se, além da grade inicial subdividida em zonas, pares de células representando os limites entre zonas (paredes) e a indicação se entre essas células existe uma porta. Na Figura 3.12 é possível ver uma representação do resultado desta etapa.

3.3 Avaliação dos resultados

Durante o processo de geração existem condições que resultam em falhas, podendo ser erros ligados à configuração ou, o mais frequente, à não satisfação da conectividade entre as zonas. No caso de falha de uma execução, o algoritmo é executado N_t vezes, na tentativa de criar um resultado válido. Adicionalmente, executa-se o algoritmo outras N_s vezes para criar amostras e escolher os melhores resultados. A ideia de criar várias

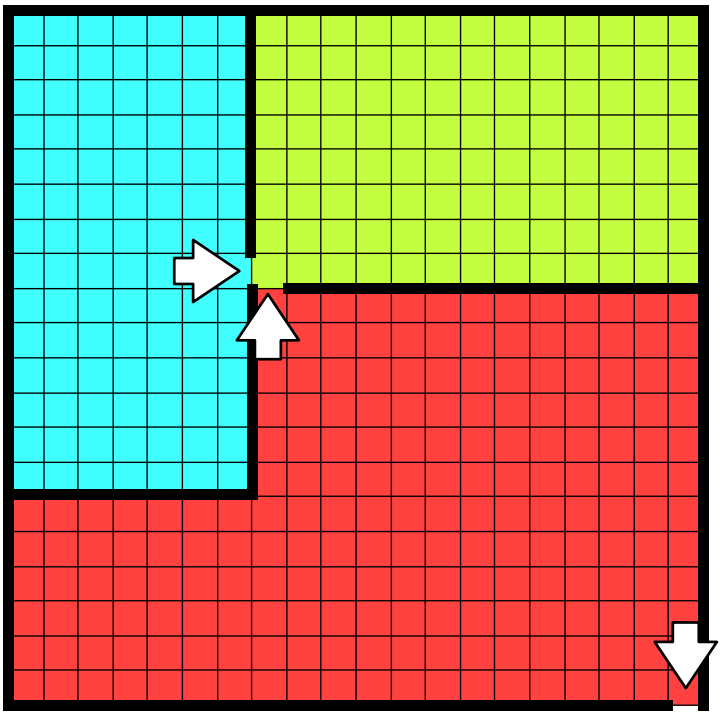


Figura 3.12: Resultado da etapa de teste de conectividade e criação das tuplas de paredes. As linhas pretas mais espessas representam paredes. As posições das portas são apontadas pelas setas brancas. Fonte: Do autor.

amostras é uma tentativa de resolver o problema de se obter resultados válidos, mas ainda assim ruins do ponto de vista prático, com zonas extremamente estreitas ou pequenas, como um quarto com formato de corredor, ou uma cozinha com somente uma célula de área.

Lopes et al (2010) sugerem o uso de características como corredores menores ou menor número de cantos. Neste trabalho, utiliza-se duas características similares às propostas e uma extra. Para corredores menores, analisa-se a razão de aspecto para evitar zonas distantes da razão definida. Para o número de cantos, faz-se a contagem de zonas retangulares. Quanto mais zonas retangulares, menos cantos. Adicionalmente, analisa-se a área final das zonas, para amenizar o problema de zonas muito pequenas, com área final muito distante da desejada. Cada amostra de planta gerada recebe uma pontuação obtida a partir da análise de cada uma dessas características.

A pontuação para o número de áreas retangulares é obtida pela Equação 3.7:

$$S_q = \frac{n_r}{n_t}, \tag{3.7}$$

onde se soma o número de zonas folhas retangulares n_r e dividindo pelo número total de zonas folhas n_t, obtendo um valor entre 0 e 1, em que quanto maior for o número de zonas retangulares, mais próximo de 1 será S_q. Para a pontuação do aspecto, é usada a Equação 3.8:

$$S_r = 1 - \frac{\sum_{i=1}^{n_t} |r_d - r_a|_i}{n_t}, \tag{3.8}$$

onde a soma dos módulos das diferenças entre o aspecto atual r_a de cada zona folha e o aspecto desejado r_d é dividida pelo número total de zonas folhas n_t. Em seguida, subtrai-se esse valor de 1, obtendo um valor entre 0 e 1 em que quanto mais próximo de 1, mais próximas em média as zonas estão do aspecto desejado. Por fim, para calcular a pontuação da área é utilizada a Equação 3.9:

$$S_a = \frac{\sum_{i=1}^{n_t} \min\left(1, \left|\frac{a_{z_i}}{r_{a_i} \cdot a_t}\right|\right)}{n_t}, \tag{3.9}$$

onde a soma das razões entre a área atual a_z e a área desejada das zonas, dada por $r_a a_t$ (onde a_t é a área total da grade), é também dividida pelo número total de zonas folhas.

A zona com maior pontuação total S_t, obtida pela soma de S_q, S_r e S_a, é escolhida e adicionada a uma lista de resultados válidos. A lista criada com os melhores resultados é então salva, e retornada para ser interpretada pelo visualizador na cena da Unity.

Essa análise, entretanto, não garantiu que todos os resultados escolhidos tivessem boa qualidade. Em algumas execuções, certas zonas terminavam com áreas muito pequenas ou alongadas, já que na média possuía valores melhores que outros resultados com várias zonas somente um pouco fora do valor esperado. Uma melhoria futura possível seria ter um limiar para excluir resultados em que ao menos uma zona tenha pontuação em alguma das características abaixo desse limiar. Isso aumentaria o número de resultados descartados, mas poderia melhorar a qualidade da geração.

3.4 Visualização dos resultados

Para visualizar os resultados da geração, foi criado um interpretador base que, além de exibir o resultado, também permite acompanhar a geração passo a passo, visando facilitar

os testes e identificar erros. Como o foco do trabalho é a geração de plantas baixas, este interpretador básico não é capaz de exibir uma construção completa com múltiplos andares, janelas e decoração. O foco foi na exibição das diferentes zonas finais geradas com suas paredes e portas. Entretanto, todo o sistema foi planejado para suportar expansão e a futura geração de construções completas, adicionando os elementos mencionados.

O interpretador utiliza uma planta baixa armazenada no resultado da geração para criar instancias de elementos gráficos em uma cena na Unity. Os elementos gráficos seguem um padrão de *assets* modulares que funcionam muito bem em mapas subdivididos em uma grade, como os apresentados na Figura 2.7. O uso desse tipo de conteúdo confere liberdade a artistas para criar construções com a estética ideal para o projeto, sem necessidade de modificações profundas no interpretador. Um interpretador que crie toda a geometria da construção também de forma procedural pode requerer modificações a cada mudança estética entre projetos, além da necessidade de um desenvolvedor com conhecimentos avançados em computação gráfica.

Esses *assets*, que aqui chamaremos de módulos, são armazenados em objetos configurados pelo usuário. Inicialmente somente 3 tipos de módulos são utilizados: um para representar as células da grade ou o piso (solo), um para paredes simples, e um para paredes com porta. Os módulos que representam o piso são posicionados a partir da leitura das células da planta, utilizando seus índices para determinar a posição no ambiente tridimensional. Os módulos de paredes e portas são posicionados a partir da leitura da estrutura de tuplas. Se a tupla possui uma porta, é criado o módulo de porta, caso contrário, é criado um de parede simples. A Figura 3.13 mostra o resultado obtido para uma planta de casa popular, com sala, cozinha, banheiro e dois quartos. A Figura 3.14 exibe a mesma planta em diferentes estilos e perspectivas, deixando clara a flexibilidade de usar módulos pré-fabricados.

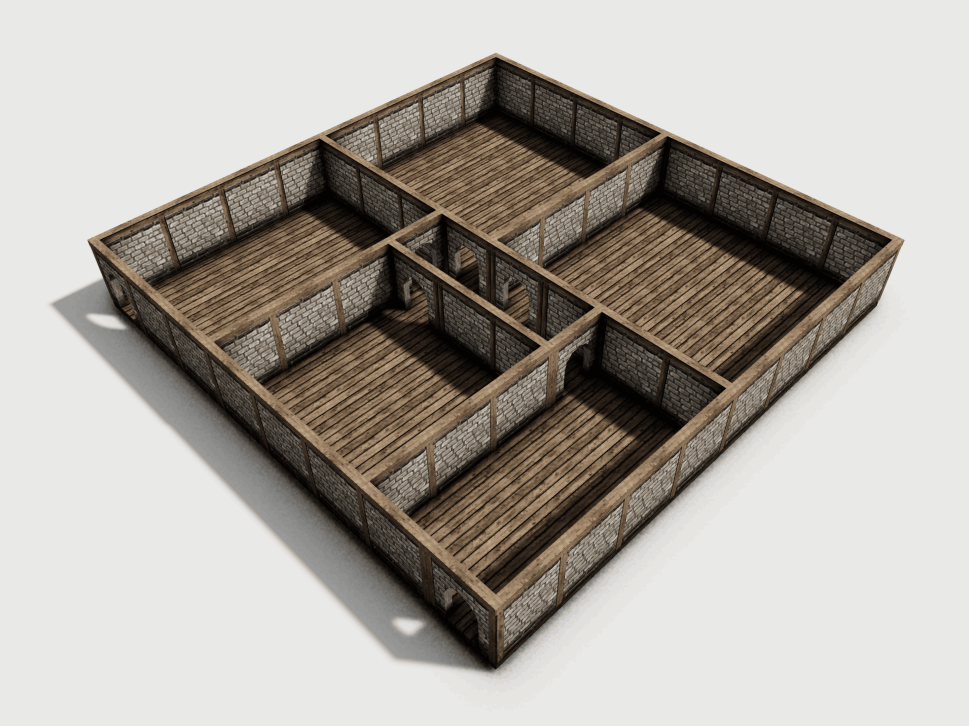
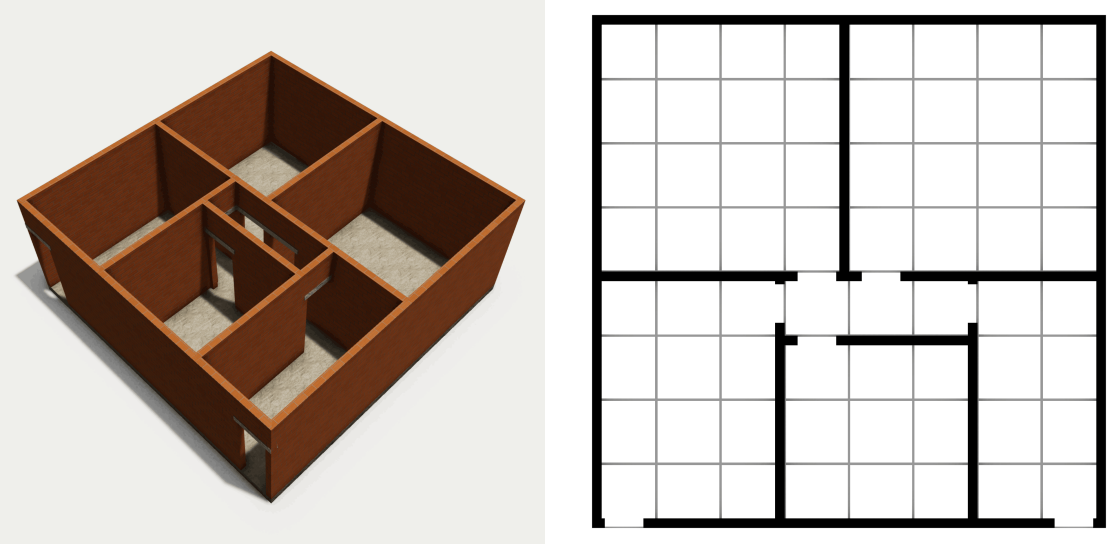


Figura 3.13: Resultado obtido utilizando o pacote apresentado na Figura 2.7. Fonte: Do autor.



(a) (b)

Figura 3.14: Em (a) tem-se uma representação tridimensional utilizando modelos feitos pelo autor deste trabalho, enquanto em (b) uma representação mais esquemática e bidimensional é apresentada. Fonte: Do autor.

4 Avaliação

Neste capitulo são apresentados alguns testes com diferentes configurações de plantas baixas para demonstrar as capacidades da ferramenta. Em Lopes et al (2010), não são apresentados dados analisando a capacidade do algoritmo de gerar resultados adequados. Os autores apenas mencionam brevemente alguns tempos de execução. Neste trabalho, para cada teste, são analisados a taxa de erro e o tempo de execução médio para criação de 100 amostras. Todos as execuções foram realizadas usando as mesmas configurações gerais do algoritmo, em uma máquina com uma CPU AMD Ryzen 5 7600x, GPU NVIDIA GeForce RTX 4060, 32 GB de memória e armazenamento SSD NVMe.

4.1 Caso de teste 1

O primeiro caso de teste é extremamente simples, equivalente a uma quitinete. A configuração possui grade com dimensão 8x8 e é dividia em 1 quarto, 1 banheiro e 1 sala.

Este caso simples, por utilizar uma grade pequena e regras simples, executou a geração de 100 plantas em 40,74 milissegundos em média, com taxa de plantas inválidas de 3,01%. Um resultado desta configuração pode ser visto nas Figuras 4.1 e 4.2, e variações na Figura 4.3.

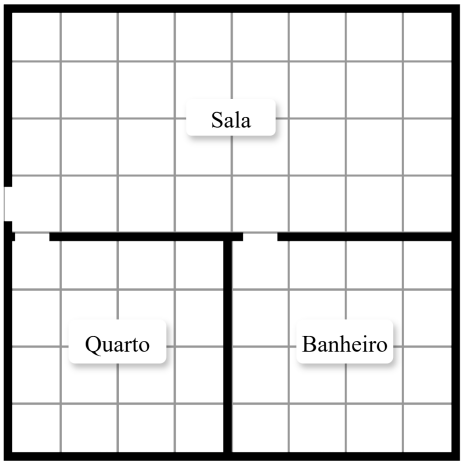


Figura 4.1: Visualização esquemática para o caso de teste 1. Fonte: Do autor.

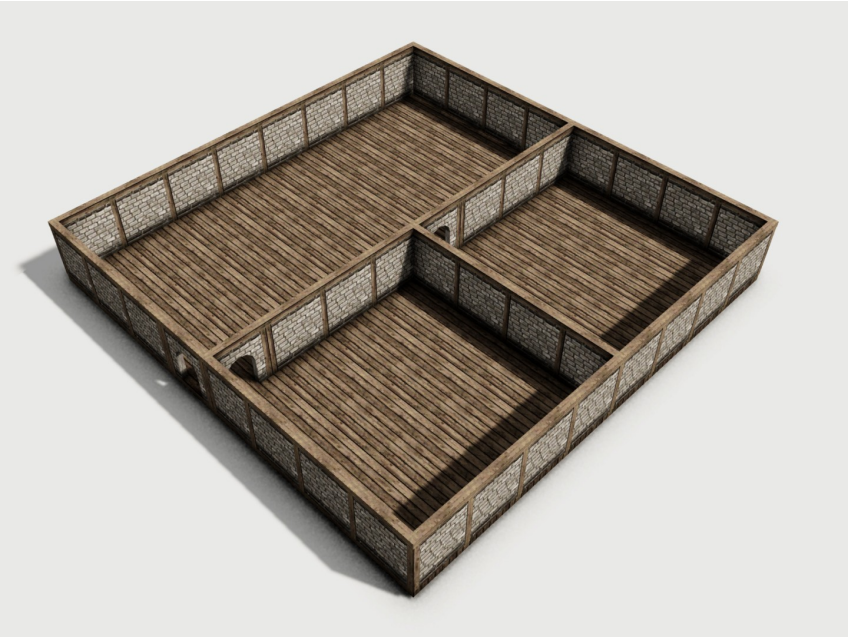
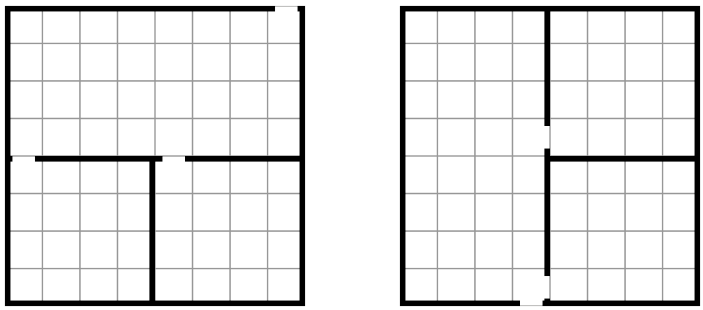


Figura 4.2: Visualização tridimensional para o caso de teste 1. Fonte: Do autor.



(a) (b)

Figura 4.3: Variações para o caso de teste 1. Fonte: Do autor.

4.2 Caso de teste 2

O segundo caso de teste é baseado em um projeto de casa popular citado em Yamanaka et al (2012). A configuração possui grade com dimensão 8x9, 2 quartos, 1 banheiro, 1 sala e 1 cozinha.

Este caso, com complexidade maior que o primeiro e pouco maior em tamanho, gerou 100 plantas em 72,48 milissegundos em média, com taxa de plantas inválidas de 73,17%. Já se nota aqui um aumento considerável na taxa de falhas. Uma possível explicação pode ser o número de zonas a serem criadas no espaço disponível. Segundo Lopes et al (2010), há chance de um cômodo crescer entre dois outros que deveriam ser

adjacentes. Assim, quanto mais zonas, maiores as chances de alguma ser posicionada de forma a gerar uma falha ligada a isso. Um resultado dessa configuração pode ser visto nas Figuras 4.4 e 4.5, e variações na Figura 4.6.

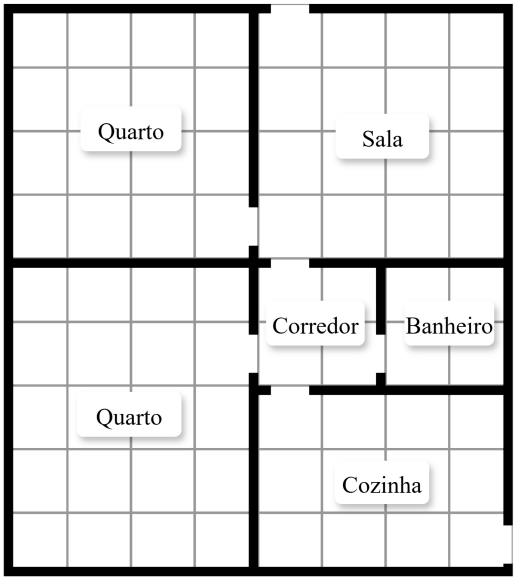


Figura 4.4: Visualização esquemática para o caso de teste 2. Fonte: Do autor.

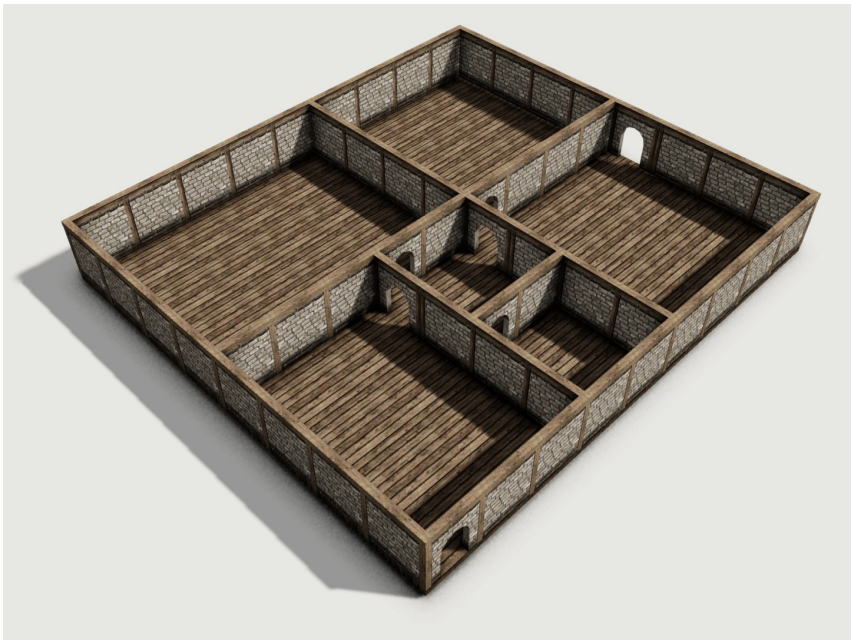
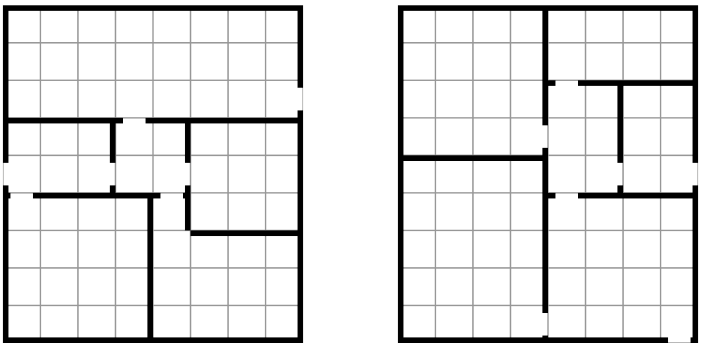


Figura 4.5: Visualização tridimensional para o caso de teste 2. Fonte: Do autor.



(a) (b)

Figura 4.6: Variações para o caso de teste 2. Fonte: Do autor.

4.3 Caso de teste 3

O terceiro caso é uma reprodução de um dos resultados mais complexos do trabalho de Lopes et al (2010), que pode ser visto na Figura 2.8. Esta planta possui uma área inicial predefinida em forma de L, uma grade com dimensão 15x15 e 9 cômodos no total, distribuídos dentro de 3 setores, sendo 2 íntimos (privados) e 1 público. O resultado pode ser visto nas Figuras 4.7 e 4.8. Variações de resultado para este caso podem ser vistas na Figura 4.9.

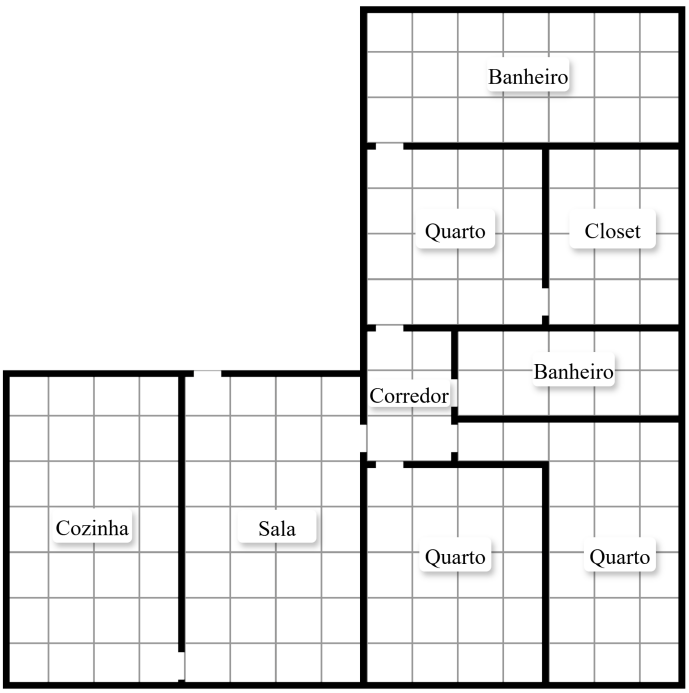


Figura 4.7: Visualização esquemática para o caso de teste 3. Fonte: Do autor.

Neste caso, com uma grade de dimensão consideravelmente maior que as anterio-

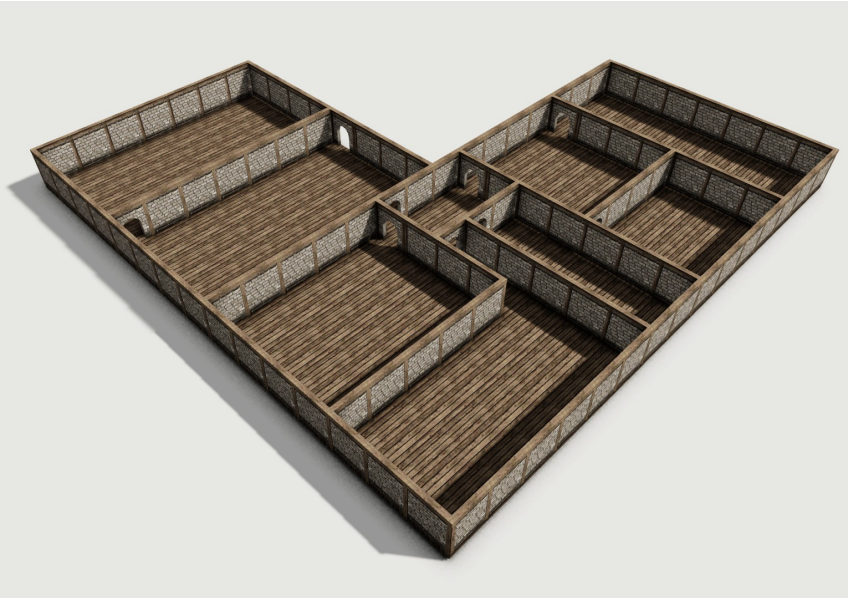
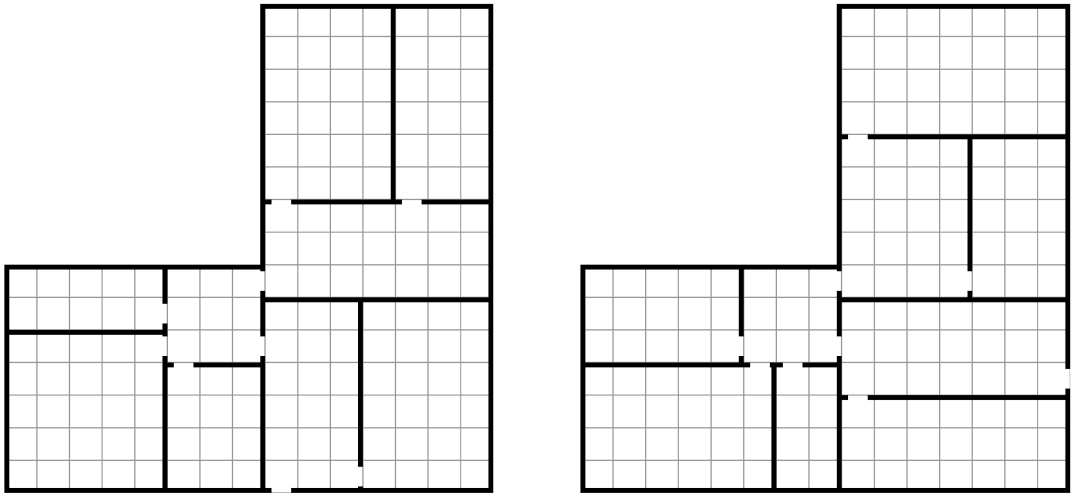


Figura 4.8: Visualização tridimensional para o caso de teste 3. Fonte: Do autor.



(a) (b)

Figura 4.9: Variações para o caso de teste 3. Fonte: Do autor.

res, nota-se um aumento significativo no tempo de processamento, atingindo agora a casa das centenas de milissegundos, com 244,4 milissegundos para criar 100 amostras, e uma taxa de resultados inválidos de 92,02%.

Na Figura 2.8a é possível ver o quarto suíte conectado à sala no trabalho de Lopes et al (2010). Isso não é possível de gerar com esta implementação sem a modificação da configuração da planta. No trabalho de referência é possível cômodos se ligarem a outros de uma zona tia, mas sem que essa conexão tenha sido definida pelo usuário. Para reproduzir este resultado, mantendo a generalização da implementação e a possibilidade

de conexão entre cômodos dentro de zonas diferentes, é possível adicionar uma regra de conectividade opcional entre zonas de níveis superiores.

4.4 Caso de teste 4

O caso de teste 4 traz uma planta com uma das subdivisões da zona raiz já definida em uma grade de 25x25 células. A tentativa deste caso é emular um piso de prédio de apartamentos com um corredor central ligado a 4 apartamentos. Cada apartamento possui configuração igual ao nosso primeiro exemplo, com uma sala, quarto e banheiro.

Com taxa de resultados inválidos de 82,36% e tempo de execução de 1456,95 milissegundos, temos um novo salto no tempo de processamento e um resultado satisfatório em termos de forma e área geral dos cômodos. Entretanto, não se observa a simetria comum neste tipo de planta. Aqui temos outra limitação do método: plantas que exigem repetição de padrões com menos randomização dos cômodos são quase impossíveis de serem geradas, especialmente de dimensões de grades maiores e com muitas subdivisões. O resultado pode ser visto nas Figuras 4.10 e 4.11, e variações na Figura 4.12.

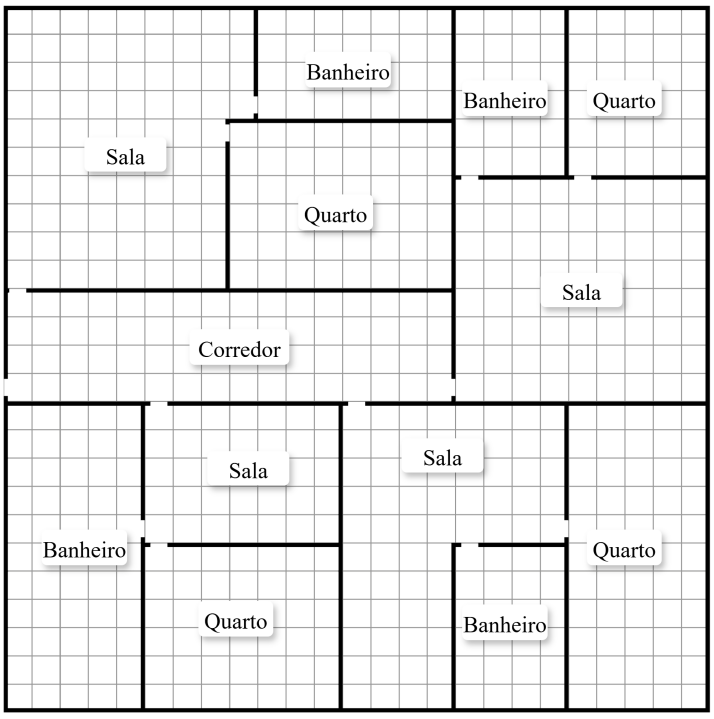


Figura 4.10: Visualização esquemática para o caso de teste 4. Fonte: Do autor.

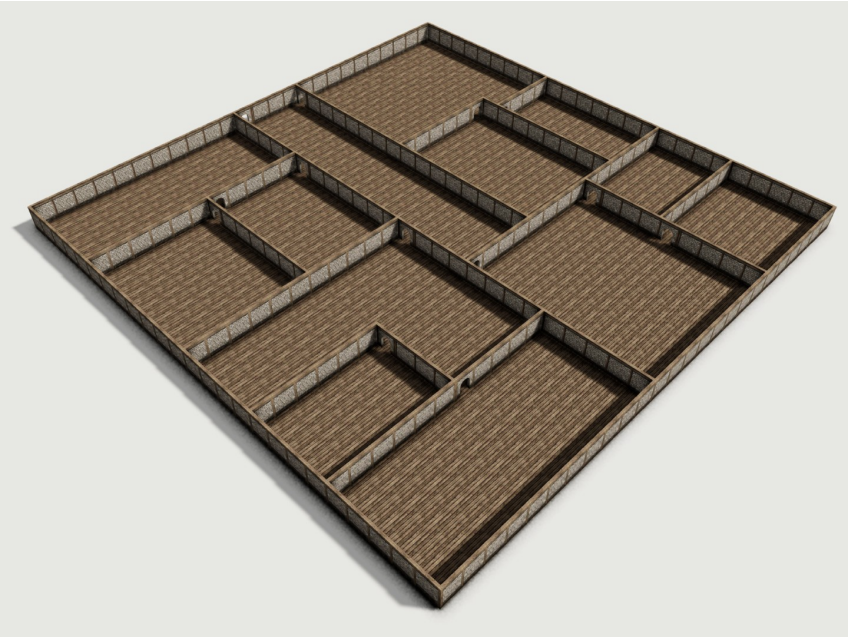
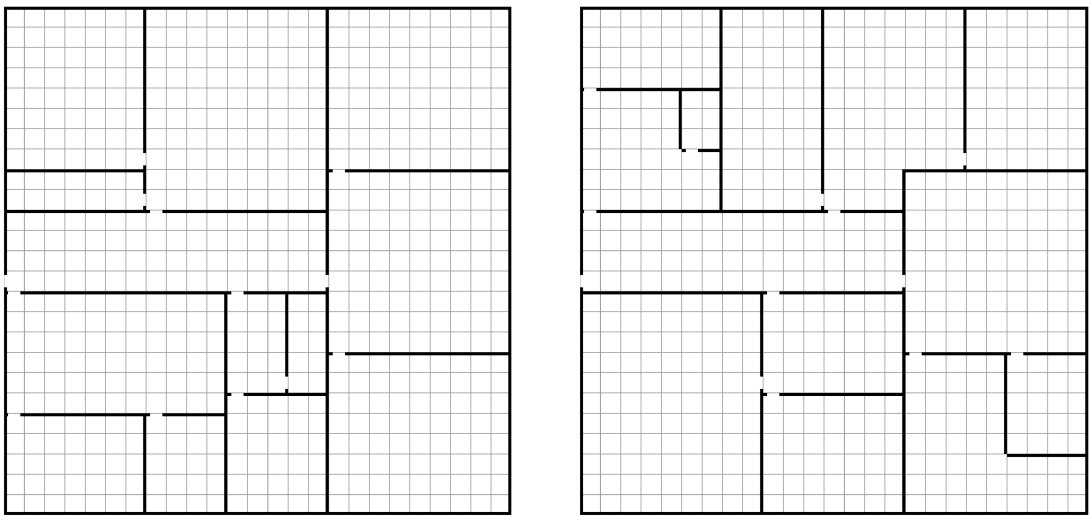


Figura 4.11: Visualização tridimensional para o caso de teste 4. Fonte: Do autor.



(a) (b)

Figura 4.12: Variações para o caso de teste 4. Fonte: Do autor.

4.5 Caso de teste 5

Este caso de teste é uma variação do caso de teste 4 com somente 3 diferenças: a grade possui 35x35 células, a área inicial da raiz é irregular, e área predefinida do corredor central foi adaptada à nova área inicial da raiz.

Com o novo aumento no tamanho da grade tivemos outro salto no tempo de processamento, atingindo agora 4314,84 milissegundos para criar 100 amostras. Entretanto, a taxa de resultados inválidos, mesmo com uma área útil irregular, não teve um

aumento muito grande em relação ao caso anterior, alcançando agora 89,32%, cerca de 7% a mais. Um dos resultados para esta configuração pode ser visto nas Figuras 4.13 e 4.14, e variações podem ser vistas na Figura 4.15.

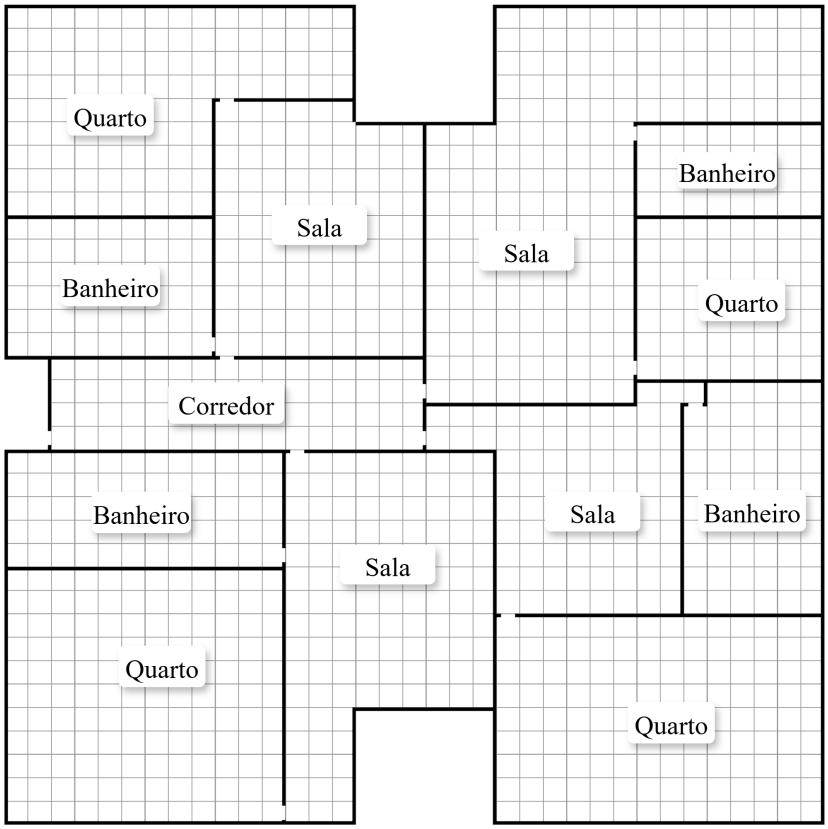


Figura 4.13: Visualização esquemática para o caso de teste 5. Fonte: Do autor.

Na Figura 4.13 é possível ver que 2 salas foram geradas com formato irregular (diferente de um retângulo ou L). Esse formato ocorre quando ao final do processo de expansão sobram células, e estas são atribuídas à zona adjacente que melhor satisfaz o critério definido pelo usuário (neste caso, a zona com área atual mais distante da área desejada).

4.6 Considerações parciais

A Tabela 4.1 reúne os resultados dos testes para os cinco casos explorados. Cada caso foi executado 100 vezes com 100 amostras, totalizando 10.000 tentativas de geração cada.

A partir dos resultados obtidos, nota-se, apesar dos resultados satisfatórios, a necessidade de melhorias que se encaixam em 3 pontos principais: a redução da taxa

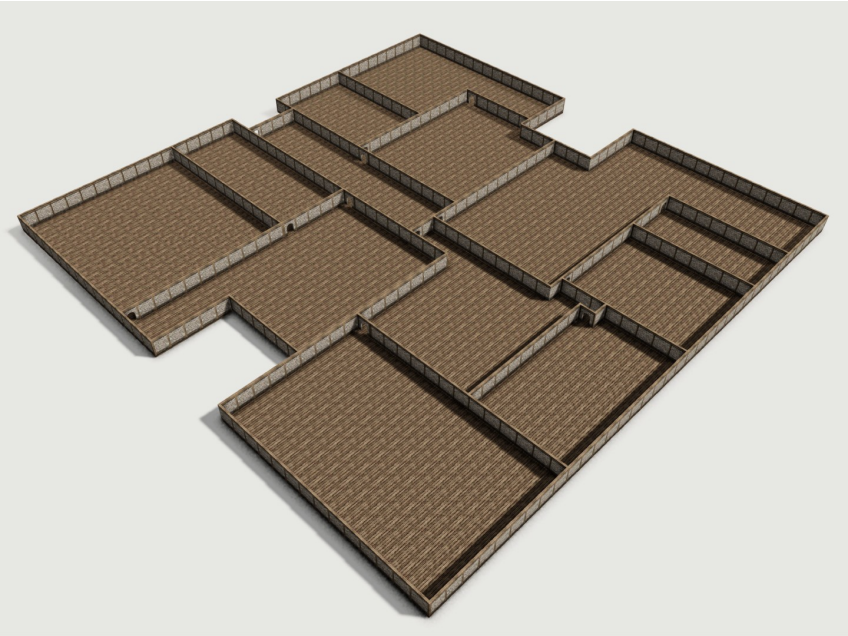
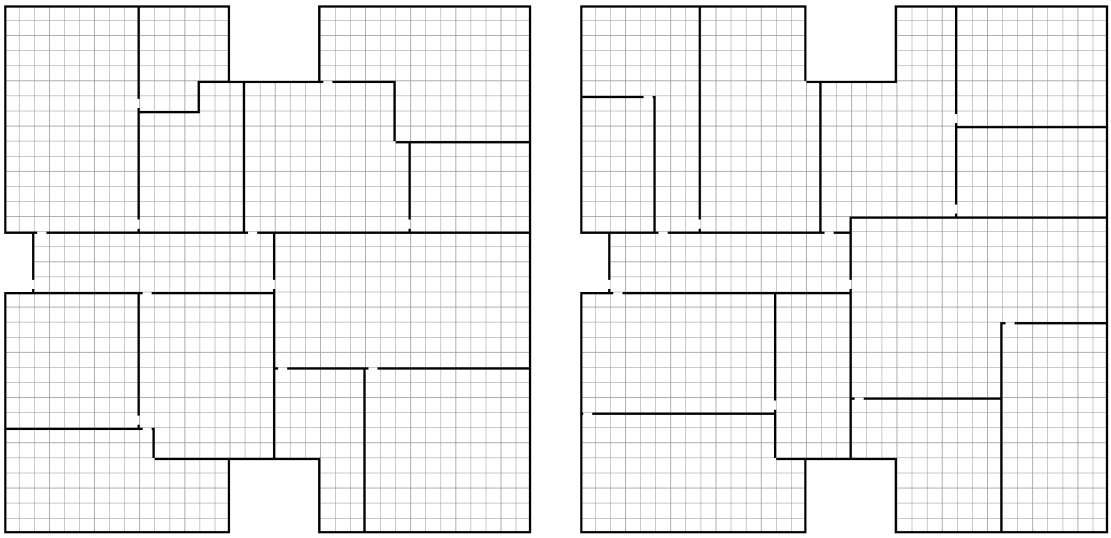


Figura 4.14: Visualização tridimensional para o caso de teste 5. Fonte: Do autor.



(a) (b)

Figura 4.15: Variações para o caso de teste 5. Fonte: Do autor.

Tabela 4.1: Resultados dos casos de teste. Fonte: Do autor.

Configuração	Dimensão da matriz	Taxa de erro	Tempo 100 amostras
Caso 1	8x8	3,01%	40,74 ms
Caso 2	8x9	73,17%	72,48 ms
Caso 3	15x15	92,02%	244,4 ms
Caso 4	25x25	82,36%	1456,95 ms
Caso 5	35x35	89,42%	4314,84 ms

de falhas, a redução do tempo de execução, melhoria da qualidade e a visualização dos resultados.

Na maior parte dos casos de teste houve mais de 80% de taxa de falha por não ser possível satisfazer as regras de conectividade. Reduzir essa taxa de falhas irá requerer menos tentativas para gerar resultados válidos, melhorando o tempo de execução. Durante os testes, verificou-se que a área desejada para cada zona também é fator determinante na taxa de erro. Zonas com muitas conexões e áreas pequenas, como corredores, podem não atingir o tamanho mínimo para satisfazer as conexões, resultando em falhas. Em alguns casos, é possível melhorar essa taxa somente modificando as configurações da planta para algo mais fácil de ser gerado. A redução da taxa de falhas pode ser obtida também por ajustes em parâmetros avançados do algoritmo, como a curva de peso da Figura 3.6, além de modificações nas equações do cálculo de pesos. Como visto na seção de execução do algoritmo, a simples mudança em algumas unidades de um denominador no momento do cálculo de pesos das bordas pode afetar o resultado.

Outra forma de reduzir a taxa de erros é modificar o algoritmo para criar corredores como um tipo especial de zona, conectando zonas ou cômodos desconexos. Trabalhos como os de Camozzato et al (2015) e Mirahmadi et al (2012) adotam uma estratégia similar para garantir o acesso a todos os cômodos criados. No trabalho de Lopes et al (2010), os corredores não são criados ou expandidos de forma diferente das demais áreas, porém são conectados automaticamente a cômodos ou zonas adjacentes sem conexão explicitada pelo usuário. Uma outra opção seria permitir ao usuário especificar quais zonas podem ser usadas para a etapa de conexão automática com os corredores. Por exemplo, pode-se definir que a zona configurada como um corredor se conecte livremente a qualquer cômodo filho da zona privada e pública.

Além da redução de falhas, para reduzir o tempo de execução é possível diminuir o número de verificações durante as etapas de crescimento e cálculo de pesos, além de adicionar paralelismo. Durante o desenvolvimento foi testada a execução em paralelo para gerar múltiplas amostras, mas problemas ligados à condição de corrida exigiriam mais trabalho para manter o algoritmo estável. Assim, para o escopo deste trabalho, a execução em paralelo não foi totalmente implementada. Além disso, a redução de verificações requer a reescrita de partes cruciais do algoritmo, o que não seria viável no momento.

Para melhorar a qualidade dos resultados pode-se incorporar regras de engenharia e arquitetura que levam em conta questões estruturais, de circulação de pessoas e distribuição de setores. Estas regras devem ser preferencialmente opcionais para abranger construções atípicas em contextos de fantasia ou ficção e não elevar a taxa de erro ao descartar resultados que seriam de baixa qualidade em um contexto realista.

Por fim, para visualização dos resultados, pode-se criar um algoritmo mais elaborado que suporte variações dos módulos, construa telhados não planos, e consiga ler múltiplas plantas para criar construções com mais de um piso. Além disso, a possibilidade da adição de mobílias e elementos decorativos de forma procedural também seria uma grande melhora. Esses elementos a serem adicionados não precisam se limitar a *assets* tridimensionais, podendo também ser adicionados elementos como luzes e *scripts* com comportamento customizado. Um exemplo é a criação de portas e interruptores de luz interativos.

5 Considerações finais

Este trabalho apresentou um estudo sobre técnicas para geração procedural de plantas baixas e a implementação de um algoritmo que atendesse aos requisitos especificados no capítulo introdutório. Foi realizado um levantamento da bibliografia associada ao tema, com vistas a identificar o algoritmo que seria mais adequado ao objetivo do trabalho. O método de Lopes et al (2010) foi o escolhido por sua relativa simplicidade de implementação e por servir de base a outras derivações posteriores. O algoritmo foi testado com cinco casos de teste, abrangendo desde plantas simples até estruturas complexas envolvendo vários cômodos, e os resultados do algoritmo foram analisados através das porcentagens de falha obtidas para um conjunto fixo de amostras geradas. No geral, os resultados foram plausíveis, apesar das taxas de falha terem sido um pouco elevadas para boa parte dos casos.

A implementação feita do algoritmo escolhido atendeu a maioria dos requisitos propostos. Por ser de fácil implementação, foi possível chegar a resultados similares ao de Lopes et al (2010), apesar de o trabalho original não apresentar alguns detalhes de implementação e soluções para certos problemas encontrados no desenvolvimento. O potencial para expansão, melhorias e integração foi identificado em diferentes partes, principalmente na melhora de qualidade dos resultados e exibição.

A implementação foi feita de forma a generalizar o contexto de geração para não se limitar a construções padrão. Conforme dito anteriormente, o método de Lopes et al (2010) foca em construções típicas norte-americanas. Já na implementação realizada neste trabalho, o usuário pode ser criativo e aplicar o algoritmo para criar diferentes estruturas com diversos fins que não sejam casas ou edifícios tradicionais.

Em termos de velocidade, o algoritmo tem potencial para ser usado em aplicações em tempo real, tendo atingido um tempo de geração inferior a 1 segundo para amostras individuais na maior parte dos testes. No pior caso, para plantas mais complexas que não possam ser geradas em tempo real, a geração poderia ser executada em uma fase de pré-processamento (por exemplo, durante o carregamento de um nível de um jogo), ou mesmo

durante a execução do jogo, à medida que o jogador se desloca pelo cenário. Também tem-se alta controlabilidade, sendo possível mudar diversos parâmetros do gerador e criar entradas de plantas totalmente personalizadas, o que cria também alta diversidade de resultados.

Para o critério de credibilidade que indica o quanto um conteúdo aparenta ter sido gerado por um humano ou de forma procedural, uma análise qualitativa mais elaborada dos resultados seria necessária para se fazer uma afirmação mais precisa. Apesar dos resultados em geral parecerem plausíveis e suficientes para aplicação em jogos, este é um critério subjetivo e pode variar de acordo com o indivíduo.

O critério de confiabilidade dos resultados foi o menos satisfeito. As plantas geradas ainda requerem a avaliação final do usuário para garantir que podem cumprir sua função, além da alta taxa de falhas que podem resultar na ausência de resultados válidos. Isso, entre outros fatores, depende dos parâmetros de entrada fornecidos pelo usuário, que podem tornar impossível a geração.

Como o objetivo era a implementação de um algoritmo já existente que atendesse aos critérios mencionados, pode-se dizer que este trabalho atingiu esse objetivo. Embora o algoritmo não traga grandes melhorias em termos da qualidade dos resultados ou em desempenho, algumas pequenas variações no método original foram realizadas, dando ao usuário mais controle na geração das construções, além de ser capaz de gerar resultados similares aos do trabalho de referência. Entretanto, a proposta do sistema completo (configuração, geração e visualização) em mais detalhes e a disponibilização do código-fonte de forma pública estabelecem a base para o desenvolvimento de uma ferramenta mais robusta e acessível, o que também era um objetivo deste trabalho. Em certa medida, o objetivo de se obter uma ferramenta de utilização simples foi atingido. No entanto, melhorias de interface e validação dos dados fornecidos pelo usuário ainda são necessários para tornarem o seu uso mais simples.

O software desenvolvido ainda possui algumas limitações, como a necessidade de dezenas de execuções para encontrar uma solução válida na maioria dos cenários testados, e a necessidade de avaliação pelo usuário, visto que que nem sempre os resultados apresentam qualidade satisfatória. A geração completa de múltiplos pisos também é um desafio.

Apesar do software implementar os recursos básicos para isso, ainda são necessários refinamentos na entrada de dados e na visualização.

Como trabalhos futuros têm-se, inicialmente, a prioridade em resolver as limitações citadas e, a partir disso, pode-se realizar outras melhorias e expansão do sistema. Um novo recurso interessante é a geração também procedural de formas ou áreas iniciais para não restringir apenas às definidas pelo usuário. Um outro caminho natural seria a implementação de melhorias propostas em trabalhos posteriores ao de Lopes et al (2010), como o de Camozzato et al (2015), que adiciona novas regras de posicionamento inicial e expansão dos cômodos. Outro possível aprimoramento, que pode ser feito nas últimas etapas do algoritmo principal ou já na visualização do resultado, é a adição de novos elementos arquitetônicos como janelas.

Bibliografia

Adão, T.; Pádua, L.; Marques, P.; Sousa, J. J.; Peres, E. ; Magalhães, L. Procedural modeling of buildings composed of arbitrarily-shaped floor-plans: Background, progress, contributions and challenges of a methodology oriented to cultural heritage. **[S.l.]: Computers**, v.8, n.2, p. 38, 2019.

Camozzato, D. **A method for growth-based procedural floor plan generation**. 2015. Dissertação de Mestrado - Porto Alegre: Pontifícia Universidade Católica do Rio Grande do Sul.

Compton, K.; Osborn, J. C. ; Mateas, M. **Generative methods**. In: The Fourth Procedural Content Generation in Games workshop, PCG, volume 1, 2013.

Creations, A. **Fake interiors**, 2019. Disponível em `https://assetstore.unity.com/packages/vfx/shaders/fake-interiors-free-104029`. Acesso em 05 de ago. de 2025.

Ebert, D. S. **Texturing & modeling: a procedural approach**. [S.l.]: Morgan Kaufmann, 2003.

Flack, R. W.; Ross, B. J. **Evolution of architectural floor plans**. In: European conference on the applications of evolutionary computation, p. 313–322. [S.l.]: Springer, 2011.

Freiknecht, J.; Effelsberg, W. Procedural generation of multistory buildings with interior. **[S.l.]: IEEE Transactions on Games**, v.12, n.3, p. 323–336, 2019.

Germer, T.; Schwarz, M. **Procedural arrangement of furniture for real-time walkthroughs**. In: [S.l.]: Computer Graphics Forum, volume 28, p. 2068–2078. Wiley Online Library, 2009.

Guerrero, P.; Jeschke, S.; Wimmer, M. ; Wonka, P. Learning shape placements by example. **New York: ACM Transactions on Graphics (TOG)**, v.34, n.4, p. 1–13, 2015.

Guo, Z.; Li, B. Evolutionary approach for spatial architecture layout design enhanced by an agent-based topology finding system. **[S.l.]: Frontiers of Architectural Research**, v.6, n.1, p. 53–62, 2017.

Guzdial, M.; Snodgrass, S. ; Summerville, A. J. **Procedural Content Generation Via Machine Learning: An Overview**. [S.l.]: Springer, 2022.

He, F.; Huang, Y. ; Wang, H. **iplan: Interactive and procedural layout planning**. In: [S.l.]: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, p. 7793–7802, 2022.

Games, H. **No man's sky**, 2016. Disponível em `https://www.nomanssky.com/`. Acesso em 05 de ago. de 2025.

Hendrikx, M.; Meijer, S.; Van Der Velden, J. ; Iosup, A. Procedural content generation for games: A survey. **New York: ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, v.9, n.1, p. 1–22, 2013.

Hohmann, B.; Havemann, S.; Krispel, U. ; Fellner, D. A gml shape grammar for semantically enriched 3d building models. **[S.l.]: Computers & Graphics**, v.34, n.4, p. 322–334, 2010.

Kenney. **Retro medieval kit**, 2025. Disponível em `https://www.kenney.nl/assets/retro-medieval-kit`. Acesso em 05 de ago. de 2025.

Kutzias, D.; von Mammen, S. Recent advances in procedural generation of buildings: From diversity to integration. **IEEE Transactions on Games**, 2023.

Leblanc, L.; Houle, J. ; Poulin, P. **Component-based modeling of complete buildings.** In: [S.l.]: Graphics Interface, volume 2011, p. 87–94, 2011.

Liapis, A. **10 years of the pcg workshop: Past and future trends**. In: [S.l.]: Proceedings of the 15th International Conference on the Foundations of Digital Games, p. 1–10, 2020.

Liu, J.; Snodgrass, S.; Khalifa, A.; Risi, S.; Yannakakis, G. N. ; Togelius, J. Deep learning for procedural content generation. **Neural Computing and Applications**, v.33, n.1, p. 19–37, 2021.

Lopes, R.; Tutenel, T.; Smelik, R. M.; De Kraker, K. J. ; Bidarra, R. **A constrained growth method for procedural floor plan generation**. In: [S.l.]: Proc. 11th Int. Conf. Intell. Games Simul, p. 13–20. Citeseer, 2010.

Macri, D.; Pallister, K. **Procedural 3d content generation.** [S.l.]: Technical report, Intel Developer Service, 2000.

Marson, F.; Musse, S. R. Automatic real-time generation of floor plans based on squarified treemaps algorithm. **[S.l.]: International Journal of Computer Games Technology**, v.2010, n.1, p. 624817, 2010.

Merrell, P.; Schkufza, E. ; Koltun, V. **Computer-generated residential building layouts**. In: [S.l.]: ACM SIGGRApH Asia 2010 papers, p. 1–12. 2010.

Merrell, P.; Schkufza, E.; Li, Z.; Agrawala, M. ; Koltun, V. Interactive furniture layout using interior design guidelines. **New York: ACM transactions on graphics (TOG)**, v.30, n.4, p. 1–10, 2011.

Mirahmadi, M.; Shami, A. A novel algorithm for real-time procedural generation of building floor plans. **[S.l.]: arXiv preprint arXiv:1211.5842**, 2012.

Müller, P.; Wonka, P.; Haegler, S.; Ulmer, A. ; Van Gool, L. **Procedural modeling of buildings**. In: [S.l.]: ACM SIGGRAPH 2006 Papers, p. 614–623. 2006.

Rodrigues, N.; Dionísio, M.; Gonçalves, A.; Magalhães, L. G.; Moura, J.-P. ; Chalmers, A. **Incorporating legal rules on procedural house generation**. In: [S.l.]: Proceedings of the 24th Spring Conference on Computer Graphics, p. 59–66, 2008.

Rogers, D. F. **Procedural elements for computer graphics**. [S.l.]: McGraw-Hill, Inc., 1986.

Sanchez, D.; Solá-Sloan, J. M. ; Lozano-Inca, E. **Procedural generation of building blueprints for real-time applications**. In: [S.l.]: Proceedings of the 2010 Spring Simulation Multiconference, p. 1–4, 2010.

Shaker, N.; Togelius, J. ; Nelson, M. J. **Procedural content generation in games**. [S.l.]: Springer, 2016.

Thaller, W.; Krispel, U.; Zmugg, R.; Havemann, S. ; Fellner, D. W. Shape grammars on convex polyhedra. **[S.l.]: Computers & Graphics**, v.37, n.6, p. 707–717, 2013.

Togelius, J.; Kastbjerg, E.; Schedl, D. ; Yannakakis, G. N. **What is procedural content generation? mario on the borderline**. In: [S.l.]: Proceedings of the 2nd international workshop on procedural content generation in games, p. 1–6, 2011.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O. ; Browne, C. Search-based procedural content generation: A taxonomy and survey. **[S.l.]: IEEE Transactions on Computational Intelligence and AI in Games**, v.3, n.3, p. 172–186, 2011.

Tutenel, T.; Bidarra, R.; Smelik, R. M. ; De Kraker, K. J. **Rule-based layout solving and its application to procedural interior generation**. In: [S.l.]: CASA workshop on 3D advanced media in gaming and simulation, 2009.

Tutenel, T.; Smelik, R. M.; Lopes, R.; de Kraker, K. J. ; Bidarra, R. Generating consistent buildings: a semantic approach for integrating procedural techniques. **[S.l.]: IEEE Transactions on Computational Intelligence and AI in Games**, v.3, n.3, p. 274–288, 2011.

Willis, A. R.; Ganesh, P.; Volle, K.; Zhang, J. ; Brink, K. Volumetric procedural models for shape representation. **[S.l.]: Graphics and Visual Computing**, v.4, p. 200018, 2021.

Wu, W.; Fu, X.-M.; Tang, R.; Wang, Y.; Qi, Y.-H. ; Liu, L. Data-driven interior plan generation for residential buildings. **New York: ACM Transactions on Graphics (TOG)**, v.38, n.6, p. 1–12, 2019.

Yamanaka, E. S. **Projeto básico para construção de casas populares**. [S. l.]: Serviço Brasileiro de Respostas Técnicas, 21 mar. 2012. Disponível em `https://respostatecnica.org.br/busca/projeto-basico-para-construcao-de-casas-populares/20987/tecnica`. Acesso em 19 de ago de 2025.

Zmugg, R.; Thaller, W.; Krispel, U.; Edelsbrunner, J.; Havemann, S. ; Fellner, D. W. Procedural architecture using deformation-aware split grammars. **[S.l.]: The Visual Computer**, v.30, n.9, p. 1009–1019, 2014.

van Aanholt, L.; Bidarra, R. **Declarative procedural generation of architecture with semantic architectural profiles**. In: 2020 IEEE Conference on Games (CoG), p. 351–358. [S.l.]: IEEE, 2020.