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

Geração Procedural de Mapas de Jogos por Cadeias de Markov

Adriano Eiterer Oliveira

JUIZ DE FORA
MARÇO, 2025

Geração Procedural de Mapas de Jogos por Cadeias de Markov

Adriano Eiterer Oliveira

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Igor de Oliveira Knop

JUIZ DE FORA

MARÇO, 2025

Geração Procedural de Mapas de Jogos por Cadeias de Markov

Adriano Eiterer Oliveira

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:

Igor de Oliveira Knop
D.Sc. Modelagem Computacional

Carlos Cristiano Hasenclever Borges
D.Sc. em Engenharia Civil

Marcelo Caniato Renhe
D.Sc. Engenharia de Sistemas e Computação

JUIZ DE FORA
12 DE MARÇO, 2025

Resumo

A indústria de jogos eletrônicos alcança a casa dos bilhões de dólares, demonstrando a aceitação e capilaridade do mercado pela sociedade. Os algoritmos para geração procedural são utilizados para diminuir o custo de produção de jogos, aumentar a quantidade de conteúdo e com isso, agregar valor ao produto para os jogadores. Explorar os métodos passados e atuais é uma forma de criar conhecimento e preparar os estudantes para este mercado, mas estas iniciativas ainda são raras nas faculdades de Ciência da Computação.Este trabalho explora a criação de mapas utilizando cadeias de *Markov* em um ambiente de experimentação chamado *Procedural Content Generation Lab* (MarkovLab). Dois métodos que utilizam cadeias de Markov de ordem superior encontrados na literatura são implementados e uma interface web é desenvolvida para o ambiente. Através dela, experimentos são realizados com o treinamento e a posterior geração de mapas para jogos são avaliados qualitativamente.

Palavras-chave: desenvolvimento de jogos; geração procedural de conteúdo; cadeias de Markov.

Abstract

The electronic games industry has reached billions of dollars, demonstrating the acceptance and capillarity of the market by society. Procedural generation algorithms are used to reduce the cost of game production, increase the amount of content and, therefore, add value to the product for players. Exploring past and current methods is a way to create knowledge and prepare students for this market, but these initiatives are still rare in Computer Science faculties. This work explores the creation of maps using *Markov* chains in an experimental environment called MarkovLab. Two methods that use higher-order Markov chains found in the literature are implemented and a web interface is developed for the environment. Through it, experiments are performed with training and the subsequent generation of maps for games are qualitatively evaluated.

Keywords: game development; procedural content generation, Markov chain.

Conteúdo

Lista de Figuras

Lista de Tabelas

Siglas

GPC Geração Procedural de Conteúdo. 13, 14

MarkovLab *Procedural Content Generation Lab*. 1–4, 9, 10, 23, 25, 26, 28, 30, 32, 33, 35, 37, 41

PRNG *Pseudo-Random Number Generator*. 15

RPG *Role-Playing Game*. 13

1 Introdução

O mercado de jogos movimenta bilhões de dólares pelo mundo. Segundo Newzoo (2024), estima-se que o mercado global de jogos gerará receita de 187,7 bilhões de dólares em 2024. Somente no país, foram 2,7 bilhões de reais movimentados, colocando o Brasil como o 10º maior mercado desse setor no mundo.

O alto grau de investimento no setor se deve à grande difusão da cultura de jogos na sociedade, o que faz com que pequenas empresas ou mesmo pequenos grupos de desenvolvedores surjam pelo mundo em busca de captar uma parte deste mercado. As grandes empresas possuem um amplo orçamento para produzirem um jogo, o que permite ter uma equipe especializada em cada etapa de desenvolvimento. A ausência de um largo orçamento faz com que as pequenas empresas tenham poucos funcionários, que acumulam diversas funções ou utilizam instrumentos para aumentar a produção e serem competitivas. A geração procedural é um destes instrumentos, que é utilizada no intuito de diminuir custos de produção e duração na criação de conteúdo para seus jogos (TOGELIUS; SHAKER; NELSON, 2016).

A geração procedural pode auxiliar em diversas etapas da geração de conteúdo, como: a posição inicial de inimigos; a geração de recompensas; a geração de atributos de armas e armaduras; visuais de personagens e inimigos; geração de regiões e ambiente por onde o jogador irá passar. Um exemplo dessa última são os jogos do tipo *roguelike*. Nesses jogos, o aspecto de aleatoriedade de todos os encontros é conferido pelo uso de um conjunto de algoritmos que garantam um alto grau de variabilidade, abrindo um amplo leque de possibilidades e restrições com as quais os jogadores devem usar a inteligência e habilidade para superar os desafios e diversificar sua experiência.

1.1 Contextualização da Geração Procedural

Segundo Smith (2014), a geração procedural surgiu na década de 1980 com a necessidade de criar conteúdo para um jogo de modo a ocupar pouco espaço em um computador.

Porém, com a evolução dos computadores, a geração procedural foi sendo esquecida, pois surgiu a viabilidade de criação de mais conteúdo pré-fabricado e de qualidade com artistas gráficos. O método de desenvolvimento por via da geração procedural ressurgiu e ganhou destaque recentemente através do advento de pequenos estúdios e desenvolvedores independentes, conhecidos como *indies*.

Ainda segundo Togelius, Shaker e Nelson (2016), geração procedural de conteúdo é um termo usado para descrever a criação de conteúdo (não mecanismos) para um jogo eletrônico, criado por um software com ou sem a assistência de uma pessoa especializada da área. Descrevendo de uma forma simples, a geração procedural consiste em utilizar algoritmos para realizar a criação de mapas, armas, armaduras, itens, personagens, entre tantos outros conteúdos utilizados em um jogo e criados de forma aleatória.

Existem vários algoritmos no campo da geração procedural, dos quais os que possuem mais relevância para a construção de mapas são: partição de espaço, onde se divide o espaço sucessivamente em salas menores; *agent-based dungeon growing*, que consiste na escavação de salas e caminhos utilizando um agente; autômatos celulares, com pequenos elementos em uma grade n-dimensional com um conjunto de estados finitos e regras de transição; geração por gramáticas livres de contexto, que consistem na utilização de frases modeladas por um conjunto finito de regras recursivas que descrevem como as estruturas de grande escala são construídas a partir de uma menor escala, utilizando uma palavra individual (SHAKER et al., 2016a). Adicionalmente, e de principal interesse deste projeto, também temos o uso das cadeias de *Markov*, um método matemático estocástico, que pode ser usado para a geração procedural de conteúdo por mudança de estados com uma determinada probabilidade (HENRY et al., 2022).

Para usar a geração procedural é necessário possuir um certo controle sobre a geração do conteúdo que será desenvolvido, pois um jogo com elementos criados de forma totalmente aleatória sem fidelidade aos aspectos estruturais daquilo que é proposto pode acabar ficando monótono (GREY, 2017). Isto é, apesar das possibilidades proporcionadas pela randomização do conteúdo, o jogo pode acabar ficando repetitivo para o jogador e os elementos podem não ter relação com a sua narrativa o que pode fazer com que o jogador o abandone, pois a experiência, ao contrário de divertida e prazerosa, pode se

tornar insípida, perdendo o aspecto fundamental da experiência preconizada pelos jogos eletrônicos.

Na indústria de jogos existem muitas produtoras criando jogos de temas e de mecanismos variáveis que usam a geração procedural em vários dos elementos de seus jogos. Entretanto, do ponto de vista de ensino técnico, cada método de geração é estudado em um jogo ou protótipo *ad hoc*. A utilização de uma ferramenta de estudo para uso geral de geração procedural permite explorar os métodos sem que haja a necessidade de se criar o jogo inteiro. Desta forma, este trabalho propõe o MarkovLab, um ambiente *online* para exploração do uso das cadeias de *Markov* em um ambiente controlado, independente de motor e estilo de jogo.

Propõe-se utilizar cadeias de *Markov* em conjunto com aprendizado de máquina para controlar a previsibilidade e os resultados. Por meio do treinamento do algoritmo, é viável torná-lo mais previsível e controlável, mantendo sua capacidade de gerar resultados aleatórios. Essa integração será feita na ferramenta, possibilitando sua comparação com diferentes abordagens de sua utilização.

1.2 Objetivos

O objetivo geral deste trabalho é contribuir para o ensino de desenvolvimento de jogos, mais especificamente no estudo dos algoritmos de geração procedural de conteúdo. Ele introduz o MarkovLab para experimentar diferentes abordagens para a criação de mapas de jogo com as cadeias de *Markov*. Como objetivos específicos, podemos destacar:

1. Selecionar da bibliografia uma coleção de algoritmos baseados em cadeias de *Markov* para a geração de mapas;
2. Implementar esses algoritmos no MarkovLab, em JavaScript;
3. Avaliar os resultados de treinamento a partir de diferentes conjuntos de treinamento e a geração com números pseudo aleatórios;
4. Avaliar os resultados da geração a partir de diferentes parâmetros como dimensões e estilos de entrada;

Ao atingir esses objetivos, espera-se ter um ambiente *online* para uso acadêmico e estudos subsequentes em geração procedural.

1.3 Metodologia

O presente trabalho detalha uma pesquisa exploratória, amparada por desenvolvimento de software, para explorar métodos de geração procedural de conteúdo em jogos usando cadeias de *Markov*. Os seguintes passos foram realizados para a sua execução:

1. implementação das cadeias de *Markov*;
2. implementação de treinamento das probabilidades da cadeia de *Markov*;
3. implementação da interface do MarkovLab;
4. realização de uma análise do impacto do uso de números pseudo-aleatórios;
5. realização de uma análise empírica dos mapas gerados e de suas respectivas tabelas de dados.

Espera-se observar a partir dos dados gerados se um determinado treinamento por cadeias de *Markov* consegue ter um alto grau de benefício para o desenvolvimento de jogos e em qual grau se aproximariam dos cenários utilizados no treinamento.

1.4 Organização do Trabalho

Este trabalho está organizado em cinco capítulos. Além desta Introdução, o Capítulo 2 traz os conceitos e referências para outros trabalhos que auxiliam no entendimento dos assuntos aqui desenvolvidos. O Capítulo 3 traz o estado da arte no qual se encontram as cadeias de *Markov* no uso em jogos. O Capítulo 4 apresenta a implementação das cadeias de *Markov* e os cenários utilizados para a avaliação deste trabalho. Por fim, o Capítulo 5 conclui o desenvolvimento do trabalho e apresenta as considerações finais além das possibilidades para pesquisas futuras.

2 Fundamentação

Este capítulo apresenta os conceitos necessários para o entendimento do desenvolvimento deste trabalho. Serão tratadas na Seção 2.1 as características necessárias para o desenvolvimento de jogos eletrônicos, destacando os pontos que o diferem de um desenvolvimento de software corporativo. Na Seção 2.2 temos uma revisão sobre os tipos e algoritmos da geração procedural de conteúdo, além de vários exemplos de jogos que utilizam a geração procedural. A Seção 2.3 fala sobre métodos de geração de números pseudoaleatórios e, por fim, a Seção 2.4 traz uma visão geral das cadeias de *Markov* suas principais características e sua capacidade de geração, de especial interesse para este trabalho.

2.1 Desenvolvimento de Jogos

Vários pesquisadores já apresentaram definições sobre o que é um jogo. De acordo com Caillois (1958), um jogo possui as seguintes características: jogar não é obrigatório; é envolto em um limite de espaço e tempo definido; os rumos e os objetivos não podem ser determinados com antecedência; não cria riqueza e nem bens; estabelece novas regras; cria uma segunda realidade em busca de sair da vida real. Já Huizinga (2000) define como: uma atividade optativa fora da realidade, é exercida dentro de determinados limites de tempo e espaço, segue determinadas regras, e produz sentimentos de tensão e alegria. Concluindo, podemos definir o termo jogos como sendo uma atividade recreativa, tendo como objetivo o entretenimento e o divertimento do jogador. Eles possuem um conjunto de regras que regem a busca pelos objetivos e deveriam ser equilibrados de forma justa para ser divertido a todos os participantes.

Atualmente a área de desenvolvimento de jogos abrange diversas profissões, como o *design* de jogos, de nivel, som, a programação, a produção de arte, a escrita da história, produção, testes e *marketing*. De acordo com Sharp e Macklin (2016), *game design* é a prática de arquitetar e criar como um jogo funciona, incluindo o seu núcleo, ações, temas e, mais importante, a experiência do jogo. Existem dez ferramentas básicas para a área

de *game design* que são:

1. *Restrições*: limitações postas sobre o jogador em busca de criar um desafio sobre este;

2. *Ações diretas e indiretas*: ações realizadas pelo jogador que permitem interagir com os objetos ou com o espaço do jogo, e ações que são realizados sem o contato direto do jogador;

3. *Objetivo*: dar ao jogador um propósito para o jogo;

4. *Desafio*: os desafios de um jogo surgem tanto da dificuldade em alcançar determinados objetivos quanto das mecânicas e regras que o jogo impõe;

5. *Habilidade, estratégia, risco e incerteza*: habilidade é o domínio que um jogador possui sobre as mecânicas de um jogo. Estratégia é a capacidade deste de encontrar vários caminhos para alcançar um objetivo. Risco é o uso da aleatoriedade em um jogo e a incerteza é a imprevisibilidade do que acontecerá;

6. *Tomada de decisão e feedback*: a habilidade do jogador de decidir qual será sua próxima ação para alcançar o objetivo, acompanhada da capacidade de interpretar o *feedback* fornecido pelo jogo como resposta às suas escolhas.

7. *Abstração*: a modelagem de fenômenos em forma de um jogo físico ou digital;

8. *Tema*: a estrutura lógica de um jogo que o representa; *designers* de jogos a utilizam em busca de moldar a experiência do jogador e ajudar que eles entendam o jogo de forma mais rápida e intuitiva;

9. *Storytelling*: muitas vezes, o tema de um jogo está incorporado em sua história e esta ferramenta auxilia na construção deste, em um mundo onde os jogadores habitam no entorno de seus personagens e, via um avatar, realizam ações e decisões que levam ao desdobramento da história;

10. *Contexto de jogo*: elemento que ajuda a identificar onde e por quem o jogo será jogado, influenciando diretamente a experiência proporcionada. Isso inclui a análise do local, plataforma de lançamento (mobile, console, pc) além público-alvo.

Concluindo, atualmente os profissionais que trabalham no mercado de desenvolvimento de jogos precisam dominar as ferramentas básicas de *game design* em busca de proporcionar um jogo que seja divertido e simultaneamente desafiador.

2.2 Geração Procedural em Jogos

A Geração Procedural de Conteúdo (GPC) é a criação algorítmica de conteúdo de jogos com base em uma entrada limitada como restrições e dados básicos, ou indireta da equipe de produção do jogo. Em outras palavras, a GPC refere-se a um *software* de um computador que consegue criar conteúdo para um jogo (TOGELIUS; SHAKER; NELSON, 2016).O conteúdo que pode ser gerado de um jogo pela geração procedural é: os níveis, os mapas, as regras, as texturas, as histórias, os itens, as missões, as músicas, as armas, os veículos, os personagens, entre outros.

Um dos benefícios mais aparentes do uso de GPC é a não necessidade de utilizar um *designer* ou artista em cada um dos elementos do jogo. Assim, uma empresa que consegue redirecionar parte de seus artistas para outras tarefas, pode diminuir o custo de manutenção em manter uma equipe grande para ganhar uma vantagem sobre a concorrência, ou diminuir o tempo de produção de artes para um jogo. Mas também podemos inverter o pensamento ao criar um GPC que seja uma ferramenta para aumentar a criatividade de artistas e de *designers*, assim aumentando a produtividade ao invés de diminuir custos, criando a oportunidade de pequenas empresas poderem criar jogos ricos com grande volume de conteúdo (TOGELIUS; SHAKER; NELSON, 2016).

2.2.1 Jogos que utilizam a geração procedural

Os jogos que utilizam a geração procedural de forma mais extensa são dos gêneros *dungeon crawlers*, *roguelikes* e *survivals*. Os *dungeon crawlers* são um subgênero do *Role-Playing Game* (RPG), consistindo na exploração de níveis conhecidos como masmorras, mas popularmente já chamados de *dungeons*. Os *roguelikes* também são um subgênero do RPG, mas focam na exploração de mapas gerados proceduralmente, em que o jogador tem uma única vida: a morte do personagem é permanente e o jogo deve ser recomeçado. Já os

jogos conhecidos como *survival* desafiam os jogadores a sobreviver o maior tempo possível a partir da coleta de recursos para fabricação de itens em um ambiente hostil.

Atualmente muitos jogos que utilizaram a GPC conseguiram obter um grande sucesso no mercado. Por exemplo, *Diablo* (BLIZZARD, 1996) com a sua de geração de *dungeons* e itens fortemente ligados na história; *Minecraft* (MOJANG STUDIOS, 2009) com a sua geração procedural de terrenos usando blocos; *Borderlands* (GEARBOX SOFTWARE, 2009), jogo em primeira pessoa que utiliza a geração procedural de armas; *No Man's Sky* (HELLO GAMES, 2016) com a sua geração de biomas, planetas, sistemas solares e galáxias; *The Binding of Isaac* (HEADUP GAMES, 2011) com a sua geração de *dungeons*; *Don't Starve* (KLEI ENTERTAINMENT, 2013), jogo que consiste em sobreviver em um mapa gerado proceduralmente.

2.2.2 Métodos de geração procedural em mapas

Segundo Shaker et al. (2016b), os algoritmos mais conhecidos para a GPC de mapas são: partição de espaço, crescimento de *dungeon* por agentes, autômatos celulares e geração por gramáticas livres de contexto.

A partição de espaço consiste em subdividir um espaço bidimensional de forma sucessiva e, após essa divisão, adicionar salas ao mapa que são conectadas pelas divisões realizadas.

O método de crescimento de *dungeon* por agentes consiste em escavar salas e caminhos de forma sequencial, o que pode resultar em um mapa com aparência mais caótica.

O uso de autômatos celulares baseia-se em regras de transição e parâmetros que controlam o conteúdo gerado.

Por fim, a geração baseada em gramáticas utiliza uma estrutura de linguagem natural, que é transformada mediante um conjunto de regras pré-definidas.

2.3 Gerador de número pseudo-aleatório

Pelas características determinísticas dos algoritmos, computadores não conseguem gerar números verdadeiramente aleatórios, por isso o termo pseudoaleatório é usado. Uma semente aleatoria é um valor inicial utilizado para inicializar um gerador de números pseudoaleatórios *Pseudo-Random Number Generator* (PRNG) (SCHONS, 2023). A utilização de uma *seed* fixa permite que os experimentos sejam reexecutados sob as mesmas condições, gerando os mesmos resultados consistentes e facilitando a comparação entre diferentes algoritmos.

O uso de uma semente fixa é fundamental pelos seguintes motivos:

- **Reprodutibilidade:** Garante que os mesmos resultados sejam obtidos em diferentes execuções, desde que as condições sejam mantidas;
- **Comparação Consistente:** Permite uma avaliação justa entre diferentes algoritmos ou variações do mesmo algoritmo;
- **Facilidade de Depuração:** Auxilia na identificação de erros, ao possibilitar a repetição de cenários específicos.

Portanto, o emprego de *random seeds* não apenas garante maior controle sobre os experimentos, mas também contribui para o rigor científico e a confiabilidade dos resultados obtidos. No contexto de algoritmos baseados em geração procedural, as *random seeds* tornam-se ferramentas indispensáveis para o avanço do conhecimento e para a construção de sistemas robustos e previsíveis.

2.4 Cadeias de Markov

Cadeias de *Markov* são um modelo matemático de sistemas que realiza transições entre estados ao longo do tempo seguindo uma probabilidade para os próximos estados. Isto é, modelam transições de forma estocástica.

Uma cadeia de *Markov* é definida como um conjunto de estados $S = \{s_1, s_2, ..., s_n\}$ e a distribuição de probabilidade condicional $\mathrm{P}(P_t \mid P_{t-1})$ que representa a probabilidade de transição para o estado t dado o estado anterior $t - 1$. Um exemplo de aplicação

de cadeias de *Markov* pode ser visto através da modelagem de sentenças: uma cadeia poderia capturar a probabilidade de uma determinada palavra ocorrer, dada a palavra anterior (SNODGRASS, 2018). Neste trabalho, as cadeias de *Markov* serão divididas em três tipos: tradicionais ou padrão; de ordem superior; e as multidimensionais.

2.4.1 Cadeias de Markov padrão (primeira ordem)

Cadeias de *Markov* padrão, ou de primeira ordem, são cadeias onde a distribuição de probabilidade possui a restrição de depender apenas do estado atual e não de quaisquer estados anteriores. A Figura 2.1 apresenta um diagrama de uma cadeia de *Markov* padrão completa, onde é apresentado que o somatório das probabilidades de saída de um determinado estado tem que ser sempre 1. Cada probabilidade é representada em formato decimal. A Figura 2.1 também ilustra como é feita a matriz de transição da cadeia, onde cada linha indica a probabilidade de transição para o estado i partindo do estado j, sendo ixj respectivamente a linha e a coluna.

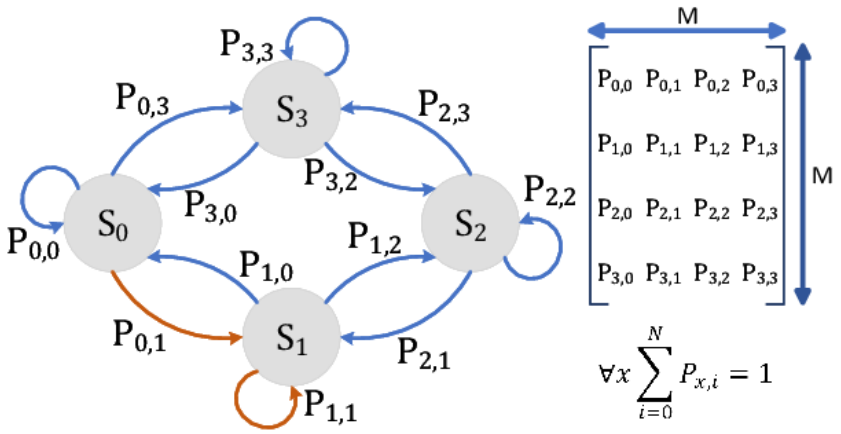

Figura 2.1: Diagrama de uma cadeia de *Markov* completa e sua matriz de probabilidade. Fonte: Salamat et al. (2019).

2.4.2 Cadeias de Markov de ordem superior

Cadeias de *Markov* de ordem superior: são cadeias onde condição de depender de apenas de um estado é relaxada, assim levando em consideração um número k de estados anteriores, onde k é um número natural finito. Em determinadas aplicações, o uso de ordens superiores permite que as cadeias de *Markov* modelem transições de estado com maior

precisão (SNODGRASS; ONTANóN, 2014b).

Esta propriedade pode ser observada na Figura 2.2, onde apresenta que uma cadeia com $k = 2$ depende de dois estados anteriores ao invés de usar apenas o estado imediatamente anterior.

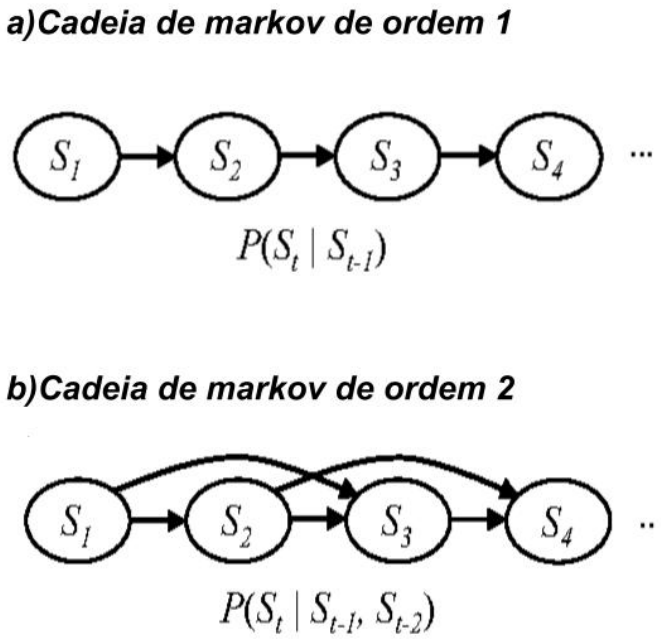

Figura 2.2: Cadeias de *Markov* ordem superior com k = 1 e 2 respectivamente. Fonte: Snodgrass e Ontanón (2014a).

2.4.3 Cadeias de Markov Multidimensionais

Cadeias de *Markov* Multidimensionais são uma extensão das cadeias de *Markov* de ordem superior que permitem representar um espaço discreto por um conjunto de estados circundantes, representados por um grafo. Assim, um determinado estado depende tanto dos vizinhos como dos estados anteriores, como pode ser visto na Figura 2.3. Na cadeia ns_3, por exemplo, cada estado depende dos três estados vizinhos, respectivamente $(l-1, t)$, $(l, t-1)$ e $(l-1, t-1)$, localizados no grafo.

Este capítulo fez uma revisão dos principais conceitos utilizados neste trabalho.

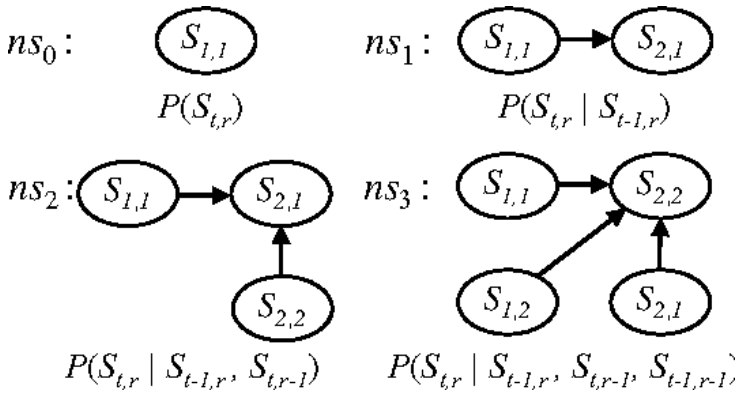

Figura 2.3: Cadeias de *Markov* multidimensionais de ordem 0, 1, 2 e 3 respectivamente. Fonte: Snodgrass e Ontanón (2016a).

No próximo capítulo, alguns dos trabalhos relacionados mencionados acima serão detalhados.

3 Trabalhos Relacionados

Este capítulo realiza uma breve revisão de artigos que utilizam as cadeias de *Markov* para a geração procedural de mapas para jogos. Uma comparação entre os artigos é realizada ao final e como eles se relacionam com este trabalho.

3.1 Geração procedural com cadeias de Markov com treinamento

No trabalho de Snodgrass e Ontanón (2016b) os autores possuem como objetivo a investigação de técnicas que podem ser utilizadas como uma base para a geração procedural de mapas independentes utilizando cadeias de *Markov*. Para a realização deste objetivo, foram utilizada técnica de treinamento que exploram os seguintes modelos da cadeia de *Markov*:

1. cadeias de *Markov* multidimensionais;
2. cadeias de *Markov* multidimensionais hierárquicas;
3. campos aleatórios de *Markov*.

Essas técnicas exigem uma coleção de mapas de treinamento, geralmente obtidos da internet. No *Markov* hierárquico, é fundamental capturar dependências de longa distância e padrões entre regiões. Isso pode ser feito manualmente, identificando e estruturando essas relações com base na observação, ou por meio de um método baseado em *clusters*, no qual os tiles de baixo nível são agrupados utilizando *k-medoids* para definir os de alto nível. Essas abordagens são aplicadas no aprendizado de máquina para avaliar a jogabilidade dos mapas gerados, conforme o gênero do jogo pretendido.

Para a realização dos experimentos, foram utilizados três jogos: *Super Mario Bros* (jogo de plataforma horizontal), *Loderunner* (jogo de labirinto), *Kid Icarus* (jogo de plataforma vertical). Para a obtenção dos dados, foi realizado um teste sobre os

mapas gerados em busca de uma porcentagem que são jogáveis, e realizado um método de comparação diferente para cada jogo, no *Super Mario Bros* e verificado se as estruturas estão completadas, no *Loderunner* verifica-se se existe um caminho que conecta todos os tesouros e o *Kid Icarus* verifica se existe um caminho entre o começo e fim do mapa.

Alcançaram como resultado que o modelo utilizando *clusters* como cadeias de *Markov* hierárquicas tiveram os melhores resultados em comparação com as outras cadeias no *Super Mario Bros*, enquanto foi pior em relação ao multidimensional e unidimensional sem o uso de *clusters*.

3.2 Cadeia de Markov com treinamento para geração e com pós-processamento

No trabalho de Lu et al. (2019) também foram utilizados os modelos de cadeias de *Markov* multidimensionais, mencionados anteriormente, em busca de gerar mapas para o Zelda (NINTENDO, 1986). Foi utilizada a mesma abordagem proposta por Snodgrass e Ontanón (2016b) para a implementação do treinamento dos modelos.

Diferente do artigo anterior, foi proposta uma representação e visualização dos mapas em multicamadas em busca de representar as bordas, pois no jogo Zelda (NINTENDO, 1986) existem *tiles* especiais que indicam a parede ou a saída. Além disso, foi proposto, após a geração do mapa, realizar-se um pós processamento onde funciona com base na busca por submapas que não existiam no conjunto de salas utilizadas para treinar o modelo. O método percorre o mapa e verifica cada submapa na sala gerada, sempre que encontra um submapa desconhecido, ele tenta substituir o ladrilho central do submapa por um ladrilho que tenha alta probabilidade, com base nos ladrilhos ao redor do submapa. para transformar um mapa não jogável em jogável. Foi demonstrado que o pós processamento melhora a quantidade de mapas utilizáveis.

3.3 Geração Procedural usando cadeias de Markov sem treinamento

No trabalho de Costa e Borchartt (2018) é proposta uma metodologia para geração procedural de terreno para jogos de plataforma 2D usando cadeia de *Markov*, possuindo como requisitos a identificação dos *tiles* que serão usados; e a identificação das respectivas probabilidades desses tiles serem o inicial e de mudança.

Como resultado, foi observado que as cadeias de *Markov* sem um devido treinamento prévio acabam inviabilizando os mapas gerados pelo algoritmo. Os autores demonstraram mediante figuras que o terreno sofre inconsistências nesses casos.

3.4 Geração Procedural usando cadeias de Markov com treinamento e adaptativo

No trabalho de Blaskowski (2016), são analisados dois métodos para a geração procedural. O primeiro método emprega cadeias de *Markov* multidimensionais combinadas com aprendizado de máquina para ajustar as probabilidades de geração. Nesse contexto, é implementado um processo em camadas (*layers*), onde uma camada é responsável por modelar o caminho provável do jogador, enquanto a outra define a altura dos tiles em relação à base. O segundo método adota uma abordagem adaptativa, utilizando modelos de teste público, como o *Generic Player Experience* (GPE) e o *Playlog Driven Categorization* (PDC), para refinar a experiência do jogador com base em seu *feedback* durante o processo de teste.

Para a realização do experimento, foram utilizados conjuntos de mapas do *Super Mario Bros*(NINTENDO, 1985), utilizando-se como uma das métricas de avaliação a posição de trampolins gerados no mapa, além da dificuldade e distância até o objetivo.

Alcançou-se como resultado uma clara distinção entre as cadeias com uma única camada e com múltiplas em relação ao trampolim, que demonstrou que o primeiro possui inconsistência sobre a utilização deste em alcançar o objetivo, enquanto o segundo é altamente consistente.

O primeiro método consegue modelar bem os dados de treino, capturando detalhes importantes, o que permite criar níveis com interações interessantes entre o jogador e o ambiente. O segundo método, por sua vez, cria níveis adaptados ao estilo de jogo de cada jogador, tornando a experiência mais personalizada.

3.5 Geração procedural usando cadeias de Markov com a utilização de agentes

No trabalho de Zafar, Irfan e Sabir (2019) foi utilizado o *GVG-LG framework*, uma plataforma criada com a finalidade de que os pesquisadores possam testar seus métodos contra uma variedade de jogos diferentes usando a linguagem *VHDL*. Para a geração de níveis usando as cadeias de *Markov*, um nível no *VHDL* é representado como uma matriz 2D.

O método de aprendizagem consiste em utilizar cadeias de *Markov* multidimensionais no processo de aprendizagem. Os níveis usados no processo de aprendizagem são divididos em fácil, médio e difícil, classificados conforme o número de *sprites* fornecidos pelo gerador de sementes aleatórias. Foram utilizados dois métodos de aprendizagem: Contagens Absolutas e Estimativa de Probabilidade.

O experimento proposto consistiu em utilizar uma semente aleatória que especifica como entrada o número de *sprites*, sendo garantido que não haja níveis não jogáveis utilizados no processo de aprendizagem. Para a realização dos experimentos, foram utilizados o jogo Zelda (NINTENDO, 1986), e como métrica avaliativa a porcentagem de vitória.

Para a avaliação do mapa gerado, foram utilizados vários agentes de jogo como *SampleMCTS*, *SampleOneStepLookAhead*, *doNoth-ing*, *SampleRS*, *OLETS* e *Number27*, alguns sendo melhores do que outros segundo a sua classificação no *GVG-AI* (PEREZ-LIEBANA et al., 2019 apud ZAFAR; IRFAN; SABIR, 2019).

Seguindo a hipótese do *Relative Algorithm Performance Profile*, para um nível ser bem projetado, o desempenho de um agente deve ser relativo à sua classificação. O resultado alcançado seguiu a hipótese mencionada, o que demonstra que o método de

geração produz níveis com qualidade.

3.6 Avaliação da aprendizagem em diferentes estruturas de dados

O trabalho de Summerville et al. (2018) apresenta vários experimentos feitos com os algoritmos de geração procedural e utilizando diferentes métodos de aprendizagem de máquina, com diferentes representações de conteúdo que são sequências, grades e grafos.

O resultado obtido indica que algumas abordagens baseadas em grafos podem demandar mais tempo à medida que a complexidade dos dados aumenta, enquanto a abordagem com grades tende a ocupar mais espaço na memória. Além disso, certos métodos exigem mais tempo de treinamento do que outros, e a escolha da representação mais adequada pode variar de acordo com o gênero do jogo.

3.7 Conclusões Parciais

O trabalho de Summerville et al. (2018) conclui que, ao utilizar o aprendizado de máquina em busca de treinar diferentes estruturas, dependendo da sua representação, pode melhorar ou piorar o método, o que o torna uma boa observação para o trabalho proposto.

Já o trabalho de Costa e Borchartt (2018) demonstra que ao utilizar as cadeias sem treinamento prévio pode causar uma distorção nos mapas gerados. Isso leva aos trabalhos de Snodgrass e Ontanón (2016b) e Lu et al. (2019), que demostram que, com o devido treinamento e representação, as cadeias conseguem gerar mapas que não possuem distorção. Adicionalmente, caso algum mapa gerado seja não jogável, pode-se realizar um pós-processamento com cadeias de *Markov*.Já os trabalhos de Blaskowski (2016) e Zafar, Irfan e Sabir (2019), trazem a avaliação e métodos de *machine learning*. Dessa forma, seguindo os resultados disponíveis na literatura, este trabalho utiliza o treinamento das cadeias de *Markov* e utiliza cadeias de ordem superior enquanto se atende ao tema do jogo implementado no MarkovLab.

Na Tabela 3.1, é apresentada uma comparação entre os artigos, destacando que

a maioria utiliza aprendizado de máquina com o objetivo de alcançar resultados mais satisfatórios. Além disso, são evidenciadas as preferências em relação às representações dos mapas e as distintas abordagens de avaliação adotadas.

Tabela 3.1: Tabela comparativa entre os projetos estudados.

Nome do trabalho	Aprendizado	Cadeia de Markov	Abordagem de avaliação
Snodgrass e Ontanón (2016b)	sim	Multidimensional	Níveis jogáveis
Lu et al. (2019)	sim	Multidimensional	Níveis jogáveis
Costa e Borchartt (2018)	não	Padrão	Visual do mapa
Blaskowski (2016)	sim	Multidimensional	Visual do mapa, elementos da jogabilidade e dificuldade
Zafar, Irfan e Sabir (2019)	sim	Multidimensional	qualidade
Summerville et al. (2018)	sim	Multidimensional	Análise de duração e armazenamento
Este projeto	sim	Multidimensional	Análise Visual

Concluindo, este capítulo fez uma revisão de uma seleção de trabalhos relacionados ao uso de Cadeias de *Markov*, além de apresentar que um pós-processamento faz com que um mapa não jogável se torne jogável.

4 Desenvolvimento

Este capítulo descreve o desenvolvimento do sistema MarkovLab, projetado para gerar mapas proceduralmente com cadeias de *Markov*. Serão abordados: a sua estrutura, funcionalidades nos métodos Multidimensional, Quasi-Hierárquico (implementação parcial do hierárquico) e hierárquicos. São detalhados também os testes realizados, incluindo padrões gerados, limitações técnicas e otimizações, além da avaliação do sistema sob critérios qualitativos e funcionais para identificar melhorias.

4.1 Visão geral do MarkovLab

O MarkovLab é um sistema desenvolvido para gerar mapas 2D de forma procedural utilizando cadeias de *Markov*. Sua estrutura é composta por diversas classes principais que desempenham papéis fundamentais na construção e no funcionamento do sistema descritas a seguir:.

- **index.html**: É o documento que monta a estrutura para abrir o sistema no navegador, garantindo a integração e a exibição correta dos elementos na interface do usuário.

- **Main**: Classe que orquestra o MarkovLab, configura as cenas e integra os elementos necessários a partir do *index.html*. Também é responsável por montar a tabela de dados e garantir a correta interação entre os componentes do sistema.

- **Game**: Controla a lógica geral do jogo, coordenando a interação entre o jogador, o mapa e as cenas.

- **Cena**: Gerencia o estado e a lógica da cena atual, incluindo a interação entre os elementos do mapa e outras partes do sistema.

- **Mapa**: Este componente é responsável por controlar o desenho dos elementos no *canvas*. Ele cuida da representação lógica, do desenho visual do mapa gerado e de

seus elementos.

- **Markov**: implementa o algoritmo de *Markov* para gerar mapas e realizar o treinamento necessário. Além disso, gera dados para tabelas analíticas, incluindo probabilidades de eventos, *backtracking* de ordem e a influência do mapa inicial na geração dos mapas.

- **Asset Manager**: Gerencia todos os recursos necessários para o jogo, como imagens, sons e outros arquivos, garantindo que estejam disponíveis para os outros componentes conforme necessário.

O processo de treinamento das cadeias envolve quatro etapas principais, repetidas para todos os tipos de cadeia de *Markov* com pequenas variações conforme necessário. As etapas são descritas abaixo:

1. **Definição do mapa de treinamento**:O mapa de treinamento é escolhido pelo usuário antes do início do processo de treinamento, permitindo a personalização do ambiente conforme os requisitos específicos do sistema.

2. **Definição do Tamanho das Grades**: O tamanho de cada grade é definido pelo usuário. Em seguida, o número total de grades é calculado com base no tamanho da imagem, garantindo que a grade seja uniforme e cubra toda a área do mapa.

3. **Atribuição dos Tipos de Grade**: A atribuição dos tipos de grade é realizada de acordo com o contexto e os requisitos específicos do cenário, levando em consideração as características do ambiente e os objetivos do treinamento

4. **Treinamento**: O modelo é treinado utilizando as grades. Durante o treinamento, o modelo aprende a partir das características de cada grade.

O processo de geração do mapa está organizado em três etapas, que também são repetidas para todos os tipos de cadeia de *Markov*, adaptando com pequenas variações conforme necessário. As etapas são descritas abaixo:

1. **Definição do tipo de grade**: A grade referente ao mapa de treinamento é calculada com base no tamanho do mapa que é gerado, dependendo do tipo da cadeia de *Markov*.

2. **Calcula existência do vizinho**: pega todos os vizinhos atuais da posição que será sorteada. Se o vizinho não existir na matriz de probabilidades, a ordem da cadeia é reduzida até encontrar um que exista ou a ordem chegar a zero. Caso não seja encontrado nenhum, é utilizada a probabilidade global da cadeia.

3. **Sorteio**: sorteia o próximo elemento utilizando o valor gerado pela semente aleatória [1] na matriz de probabilidades, caso exista o vizinho, senão é utilizado a probabilidade global na busca de recuperar do erro encontrado.

4.2 Implementação das Cadeias de Markov

Neste trabalho, foram implementados três tipos de cadeias de *Markov* multidimensionais: a cadeia multidimensional; a cadeia quasi-hierárquica; e a cadeia hierárquica. Devido às semelhanças entre esses métodos, foi desenvolvida uma superclasse denominada `MarkovBase`, que agrupa as funções comuns a todas as implementações. Entre as principais funções compartilhadas, destacam-se:

- `soma()`: recebe o identificador do vizinho e do ladrilho atual, verifica se o vizinho já está presente na matriz e, caso contrário, adiciona todas os estados da cadeia associadas a ele, incrementando a contagem do ladrilho.

- `calculate()`: assegura que a soma das probabilidades de cada vizinho seja igual a 1, conforme a propriedade fundamental das cadeias de *Markov*.

- `converterImagem()`: realiza a leitura da imagem de treinamento e retorna a matriz contendo os ladrilhos extraídos.

Além disso, há funções de pós-processamento responsáveis por corrigir erros gerados durante a criação do mapa, eliminar inconsistências visuais e refinar sua estrutura. Essas funções podem ajustar regiões densamente povoadas, equilibrar a distribuição de elementos e posicionar inimigos e baús de maneira estratégica, garantindo um resultado mais coeso e jogável.

[1] Disponível em ⟨https://www.npmjs.com/package/seedrandom⟩

4.2.1 Implementação das Cadeias de Markov Multidimensionais

A implementação das cadeias de *Markov* multidimensionais envolve a definição de um conjunto de estados e a criação de uma matriz de transição que descreve as probabilidades de transição entre esses estados. No contexto da geração de mapas, cada estado representa um tipo de ladrilho, e a matriz de transição define a probabilidade de que um ladrilho específico apareça adjacente a outro, na seguinte Figura 4.1 é possível ver um exemplo da cadeia *Markov* que esta representando os vizinhos com o próximo a ser sorteado em amarelo no centro, em seguida a grade referente e suas probabilidades de mudança na seguinte ordem piso, pedra, parede e bau.

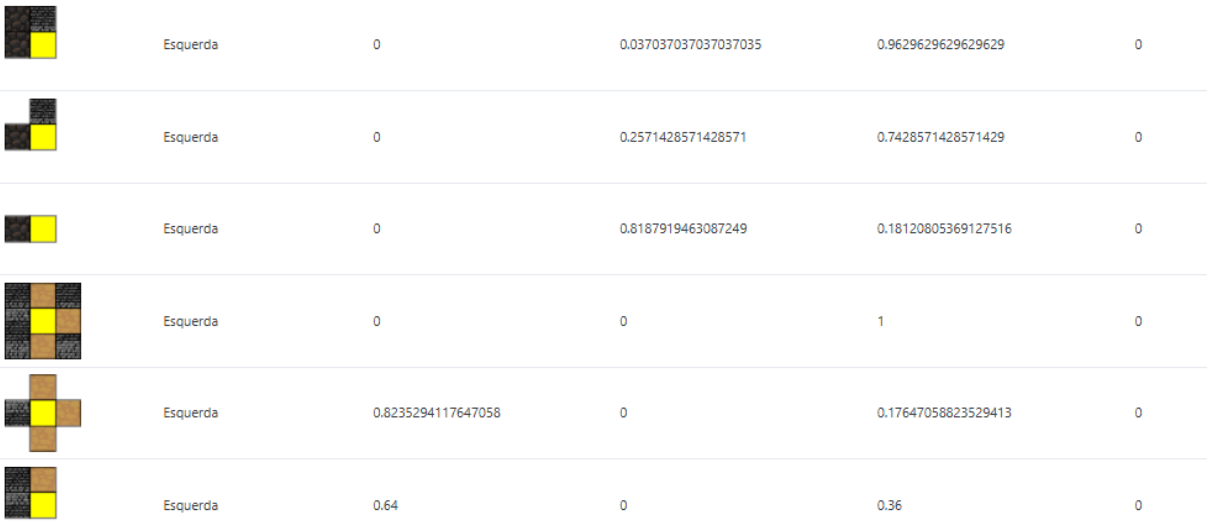

Figura 4.1: Exemplo das probabilidades do treinamento MarkovLab. Fonte: Do autor.

Essa abordagem é concretizada na classe derivada `MarkovMultidimensional`, que possui as seguintes funções principais:

- `verificaBacktracking()`: verifica se o vizinho utilizado na geração atual já está presente na matriz de probabilidades. Caso não esteja, a função reduz a ordem da cadeia, considerando que, em um mapa 2D.
- `getVizinho()`: avalia a ordem atual da cadeia e retorna o vizinho apropriado, conforme a localização do ladrilho atual, para o processo de treinamento e geração.
- `proxima()`: utiliza uma semente para determinar o próximo ladrilho a ser gerado.

A função de treinamento, denominada `treino`, atua da seguinte maneira: ela recebe a matriz obtida por meio da função `converterImagem` da classe pai e, em seguida, divide o mapa de treinamento em grades conforme o tamanho especificado na entrada. O interior de cada grade é processado para calcular as probabilidades que regem a cadeia de *Markov*. Nesta implementação, as grades não são nomeadas individualmente, sendo tratadas de forma homogênea.

4.2.2 Implementação das Cadeias de Markov Quasi-Hierarquica

Para implementar um modelo de *Markov* Quasi-Hierárquico (uma versão parcial do modelo hierárquico) inicia-se o tratamento das grades geradas durante o treinamento, aumentando assim a complexidade e a capacidade de capturar padrões mais profundos presentes no mapa de treinamento.

Essa abordagem é representada na classe `QuasiHierarquicaMarkov`. Especificamente, a função `getVizinho()` foi adaptada para receber a localização do vizinho e da grade atual, retornando o vizinho com a ordem apropriada. Além disso, na função de treinamento `treino()`, cada grade obtida na divisão do mapa de treinamento recebe um identificador numérico sequencial, assim sendo cada grade é diferente uma da outra.

4.2.3 Implementação das Cadeias de Markov Hierárquicas

Para implementar cadeias de *Markov* hierárquicas, é essencial dividir o mapa de treinamento em diferentes tipos de regiões ou padrões, como: canto; meio; e diagonais, separando estes em grades NxN durante o treinamento. Essa abordagem permite capturar e modelar as características distintas de cada região do mapa de forma mais detalhada e precisa, o que é particularmente útil em ambientes complexos que exigem uma geração procedural avançada.

Para o treinamento das cadeias de *Markov* Hierárquicas, o processo também segue as etapas mencionadas anteriormente. Antes de utilizar a matriz de ladrilhos que foi gerada pela superclasse, é necessário adicionar uma borda nesta matriz denominada `Vazio` para que assim possa realizar a geração a partir de nenhum ladrilho pré-definido, assim sendo realizada a leitura de toda a grade. Neste algoritmo, cada grade é tratada

como canto, meio, diagonais e lados. Para este fim, é utilizado o Algoritmo 1 para a seleção dos tipos durante o treinamento:

4.3 Implementação e Funcionamento da Interface Web

A implementação da interface *web* é realizada utilizando a ferramenta de estilos Pico CSS[2] e a linguagem *JavaScript* para prover interatividade e funcionalidades dinâmicas. O desenvolvimento é feito na plataforma Visual Studio, com o auxílio da extensão *Live Server* para visualização em tempo real, e o versionamento do código é gerenciado via *GitHub*.

A interface é dividida em duas partes principais: a **Entrada**, que permite ao usuário fornecer os dados necessários para a geração, incluindo a seleção do mapa inicial, o mapa de treinamento, o método de treinamento e a configuração dos parâmetros para a geração do mapa. Essa seção foi projetada para ser intuitiva e de fácil utilização, e o **Resultado da Geração**, onde, após o processamento, os resultados são exibidos, permitindo que o usuário visualize o mapa gerado, as estatísticas do processo e demais informações relevantes para a análise e validação do modelo. A interface gráfica da entrada é subdividida em:

- **Mapa Inicial**: Nesta seção (ver Figura 4.2), o usuário seleciona o mapa inicial que será utilizado para a geração, além de definir o número de linhas e colunas.

Figura 4.2: Interface de seleção do Mapa Inicial do MarkovLab. Fonte: Do autor.

- **Seleção do Mapa de Treinamento**: Nesta seção (ver Figura 4.3), o usuário seleciona o mapa que será utilizado para o treinamento das cadeias de *Markov*. Ao selecionar o mapa, sua visualização é apresentada na tela.

[2]Disponível em ⟨https://picocss.com/⟩

Algoritmo 1: Treinamento Hierárquico para Grids

```
Data: TAMANHOIMAGEM, GRID, porcentagem
Result: Matriz gi preenchida com os cantos apropriados
tamanhoGrid ← arredondarParaBaixo(TAMANHOIMAGEM / GRID);
canto ← arredondarParaBaixo(tamanhoGrid * porcentagem);
for i ← 0 to canto − 1 do
    for j ← 0 to canto − 1 do
        gi[i][j] ← "Superior esquerdo"
    end for
end for
for i ← 0 to canto − 1 do
    for j ← 0 to canto − 1 do
        gi[i][tamanhoGrid − canto + j] ← "Superior direito"
    end for
end for
for i ← 0 to canto − 1 do
    for j ← 0 to canto − 1 do
        gi[tamanhoGrid − canto + i][j] ← "Inferior esquerdo"
    end for
end for
for i ← 0 to canto − 1 do
    for j ← 0 to canto − 1 do
        gi[tamanhoGrid − canto + i][tamanhoGrid − canto + j] ← "Inferior
          direito"
    end for
end for
for i ← 0 to canto − 1 do
    for j ← canto to tamanhoGrid − canto − 1 do
        gi[i][j] ← "Cima"
    end for
end for
for i ← canto to tamanhoGrid − canto − 1 do
    for j ← 0 to canto − 1 do
        gi[i][j] ← "Esquerda"
    end for
end for
for i ← tamanhoGrid − canto to tamanhoGrid − 1 do
    for j ← canto to tamanhoGrid − canto − 1 do
        gi[i][j] ← "Baixo"
    end for
end for
for i ← canto to tamanhoGrid − canto − 1 do
    for j ← tamanhoGrid − canto to tamanhoGrid − 1 do
        gi[i][j] ← "Direita"
    end for
end for
return gi;
```

Figura 4.3: Interface de seleção do Mapa de Treinamento do MarkovLab. Fonte: Do autor.

- **Escolha do Método de Treinamento**: Nesta seção (ver Figura 4.4), o usuário seleciona o método de treinamento das cadeias de *Markov* e define o tamanho da grade que será utilizada, além da taxa de corte para o método hierárquico, assim possibilitando visualizar o resultado da divisão do mapa de treinamento.

Figura 4.4: Interface de seleção do método de treinamento do MarkovLab. Fonte: Do autor.

- **Testar**: Nesta seção (ver Figura 4.5), o usuário define o número de iterações para teste, valida as mudanças para a iteração atual e insere a string da semente que será utilizada.

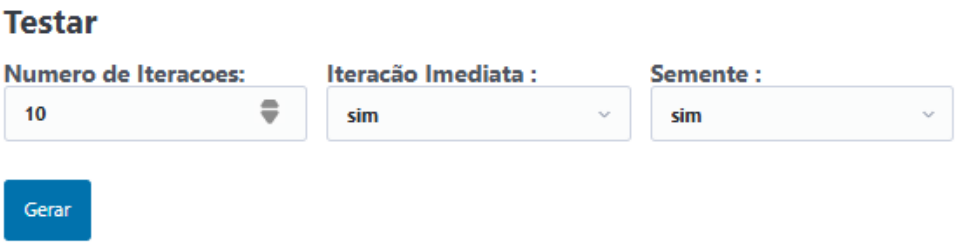

Figura 4.5: Interface para iniciar a geração do MarkovLab. Fonte: Do autor.

- **Pós-Processamento**: Nesta seção (ver Figura 4.6), o usuário define a ordem das funções de pós-processamento a serem aplicadas ao resultado da geração, permitindo a repetição e a combinação de diferentes algoritmos conforme necessário.

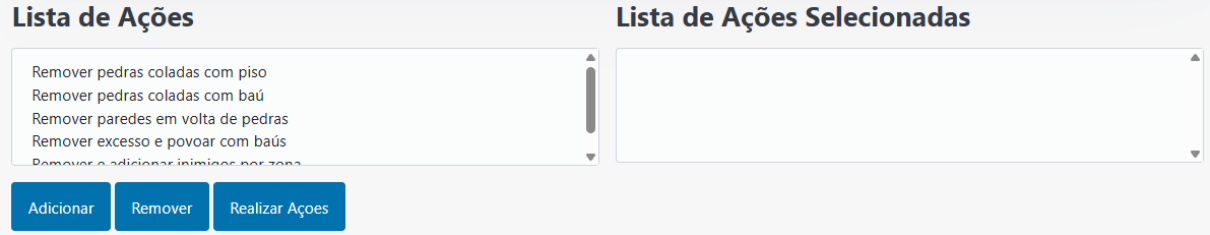

Figura 4.6: Interface para iniciar o pós-processamento do resultado alcançado pelo MarkovLab. Fonte: Do autor.

A integração entre a interface de entrada e o resultado da geração foi projetada para ser fluida, permitindo que o usuário ajuste os parâmetros conforme necessário e visualize os resultados em tempo real.

4.4 Resultados e Avaliação

Nesta seção, discutiremos os cenários de teste do sistema MarkovLab e seus respectivos resultados, analisados qualitativamente por meio de observação. Em todos os cenários, consideramos quatro tipos de ladrilhos: piso, parede, pedra e baú.

A leitura do mapa de treinamento é feita *pixel* a *pixel*, onde a cor de cada um indica o tipo de ladrilho: preto representa pedras, cinza representa paredes, amarelo indica baús, vermelho para inimigos e branco corresponde ao piso. Ao selecionar o tamanho das grades, é possível observar a separação nos modos *Multidimensionais* e *Quasi-Hierárquica* e *Hierárquica*. Nessas visualizações, as grades são destacadas em vermelho e os pixeis, em verde. No modelo hierárquico, as grades das diagonais aparecem em azul, os cantos em laranja e o centro em vermelho.

No cenário da Subseção 4.4.1, o objetivo é a geração de labirintos com corredores pequenos, enquanto no cenário da Subseção 4.4.2, a meta é atingir o simples padrão de xadrez. Já no cenário da Subseção 4.4.3, o foco é gerar diversos padrões em um único mapa.

4.4.1 Cenário 1: Exemplo simples de treinamento e geração

A Figura 4.7 mostra o mapa utilizado para o treinamento de nosso cenário, já dividido em uma grade 5×5.O objetivo é gerar um labirinto simples com corredores curtos. O tamanho deste mapa de treinamento é 10×10, o que permite a aplicação de dois métodos devido ao tamanho do mapa.

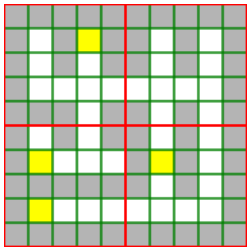

Figura 4.7: Imagem de treinamento do Cenário 1, onde as linhas vermelhas indicam a separação das grades e a linha verde representa o pixel da imagem. Fonte: Do autor.

A seguir, são listados os resultados gerados utilizando o modelo padrão como mapa inicial. A geração é iniciada a partir da linha e coluna de índice 2:

- **Resultado Quasi-Hierárquica** : A Figura 4.8 ilustra um dos resultados gerados pela cadeia de *Markov*. Nota-se que a estrutura do mapa de treinamento foi, em uma pequena parte, mantida, como pode ser visto na porção inferior esquerda, onde predominam corredores horizontais. No entanto, o mapa apresenta várias inconsistências visuais, como a presença excessiva de baús e corredores demasiadamente longos.

- **Resultado Multidimensionais**: A Figura 4.9 apresenta um dos resultados gerados pela cadeia de *Markov*. Nesse caso, a estrutura do mapa de treinamento não foi seguida, resultando em várias inconsistências visuais, como a falta de corredores em determinadas áreas e áreas com pouca conectividade.

4.4.2 Cenário 2: Padrão de Xadrez

A Figura 4.10 é utilizada na primeira parte do nosso cenário. A imagem possui o tamanho de 5×5, permitindo a mesma aplicação dos métodos do cenário anterior.

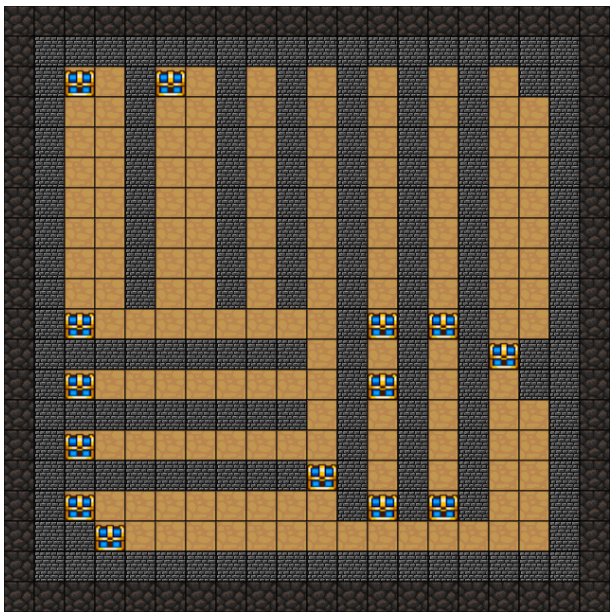

Figura 4.8: Mapa resultante do método de *markov Quasi-Hierárquica* com o uso da Figura 4.7. Fonte: Do autor.

Na geração são utilizados dois modelos de mapa inicial como pode ser visto na Figura 4.11, na primeira imagem, ao rodarmos o MarkovLab com o mapa de treinamento com grade de tamanho 5 independente do método, não é possível alcançar o modelo de xadrez proposto na imagem de treinamento devido que o modelo fica alternando em listras de pisos e paredes, devido ao mapa inicial. Na segunda imagem é possível conseguir o padrão de xadrez no resto do mapa mas é forçado devido ao começo.

A seguir, propõe-se uma alteração no mapa de treinamento, envolvendo o mapa em coberturas de pedras e paredes, conforme mostrado na Figura 4.12. Com essa mudança, o modelo 1 da Figura 4.11 frequentemente resulta em um mapa preenchido de paredes devido ao primeiro ladrilho começar com parede. No entanto, ao realizar uma iteração adicional sobre esse resultado, é possível gerar o padrão de xadrez desejado, ou como também pode ser visto na mesma figura, simplesmente inverter o começo com piso.

Com a mesma ideia de substituir a cobertura do mapa de treino e o modelo inicial por pedras, é possível alcançar o modelo proposto com apenas uma iteração, como mostrado na Figura 4.13.

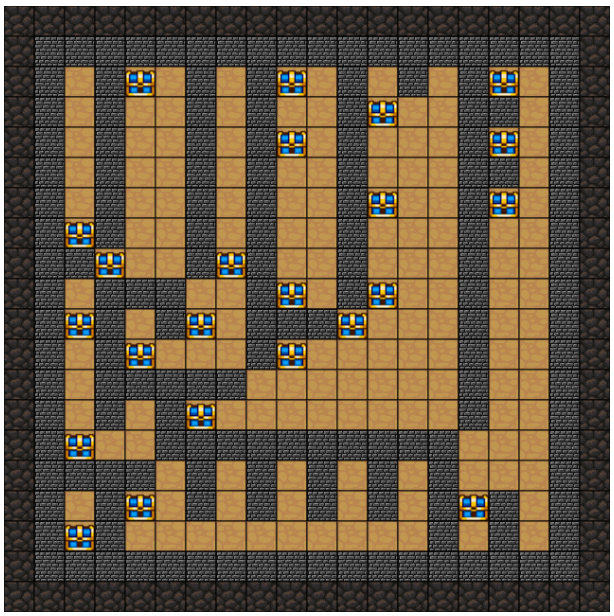

Figura 4.9: Mapa resultante do método de *markov* Multidimensional com o uso da Figura 4.7. Fonte: Do autor.

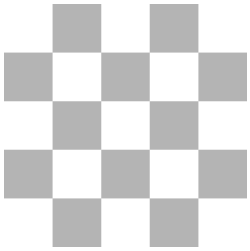

Figura 4.10: Mapa de treinamento contendo o padrão de xadrez. Fonte: Do autor.

4.4.3 Cenário 3: Padrões diversos em um único mapa

Neste cenário, nosso objetivo é buscar os padrões propostos em um mapa de treinamento de tamanho consideravelmente maior em relação aos anteriores a partir do modelo padrão da Figura 4.11.

O primeiro mapa de treinamento pode ser visto na Figura 4.14, onde são visíveis diversos padrões. Para este mapa, serão utilizados todos os métodos propostos para o treinamento. Os seguintes resultados foram obtidos com grades 5×5:

- **Markov Multidimensionais**: Não foi possível alcançar nenhum dos padrões presentes no mapa de treinamento, foram gerados apenas uma grande área aberta com alguns baús e listras de paredes espalhados, não havendo diferença com a mudança das sementes, como pode ser visto na Figura 4.15. Ao aumentar o tamanho das gra-

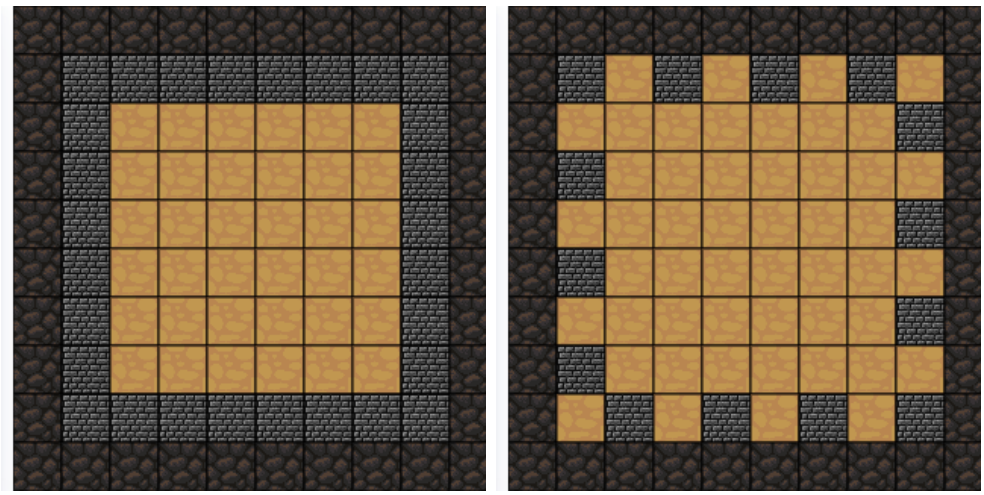

Figura 4.11: Modelo Inicial de mapa do MarkovLab o primeiro sendo o padrão, e o segundo o xadrez. Fonte: Do autor.

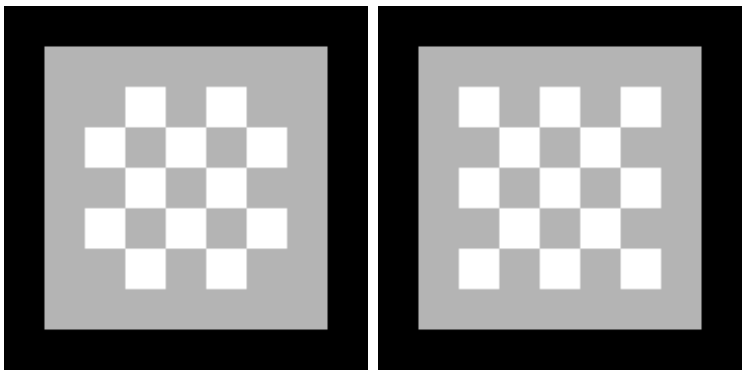

Figura 4.12: Mapas de treinamento do padrão de xadrez com bordas. Fonte: Do autor.

des, a quantidade de dados aumenta, o que faz com que os elementos predominantes do mapa de treino tenham mais chance de ocorrer.

- **Quasi-Hierárquica**: Foi o método mais eficaz para alcançar alguns dos padrões presentes no mapa de treinamento, como pode ser visto na Figura 4.16. No entanto, o resultado apresenta diversos erros visuais.

- **Markov Hierárquica**: Não é possível observar nenhum padrão distinto, como salas ou corredores, conforme mostrado na Figura 4.17. O resultado da geração foi obtido com uma única iteração e uma taxa de corte de 25 para diagonais.

Como pode ser observado, o método *Markov Quasi-Hierárquico* obteve o melhor resultado com base no mapa de treinamento. De acordo com a fundamentação, o melhor desempenho deveria ser alcançado pelo modelo hierárquico. No entanto, esse resultado

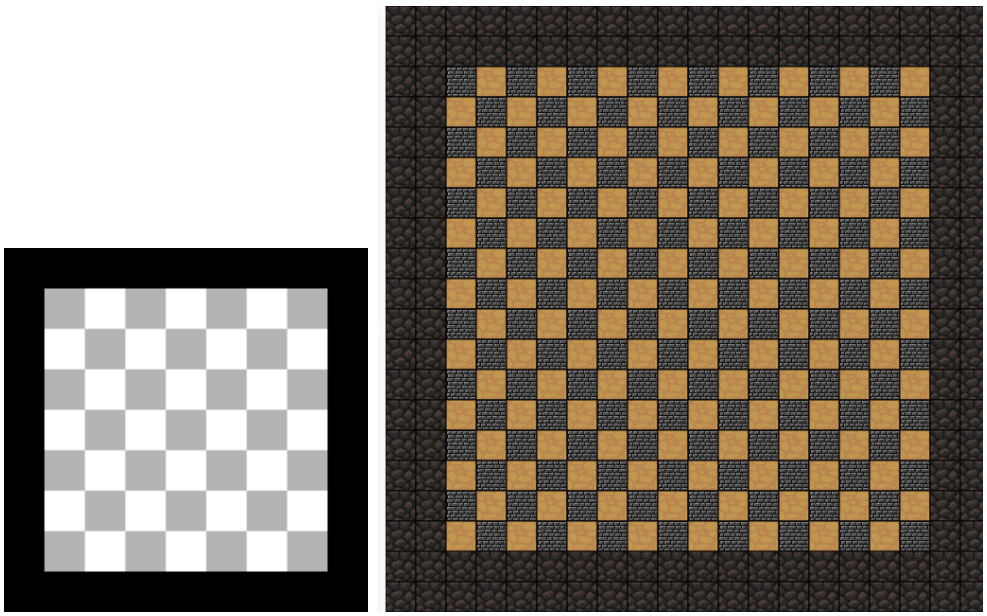

Figura 4.13: Mapa de treinamento xadrez envolta de pedra e seu respectivo resultado da geração com a mudança em uma única iteração. Fonte: Do autor.

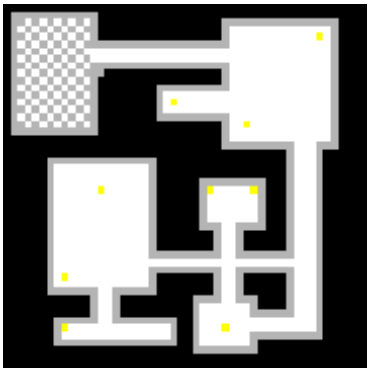

Figura 4.14: Mapa de treinamento com variações de padrões. Fonte: Do autor.

foi comprometido devido à geração não ter iniciado em um mapa vazio, como sugerido na fundamentação do método. Na Figura 4.18, é possível perceber que, ao não se utilizar um ladrilho inicial, alguns padrões foram gerados com sucesso, mas muitos outros foram perdidos devido à dificuldade em identificar corretamente os tipos de grades.

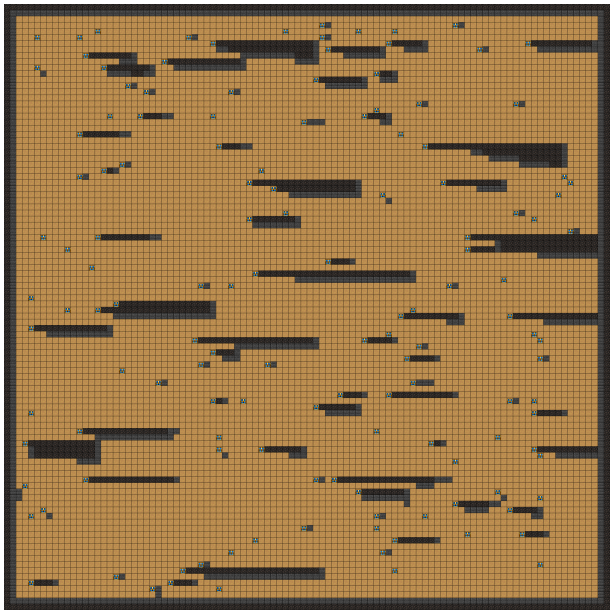

Figura 4.15: Resultado da geração de *Multidimensionais Markov* do mapa da Figura 4.14. Fonte: Do autor.

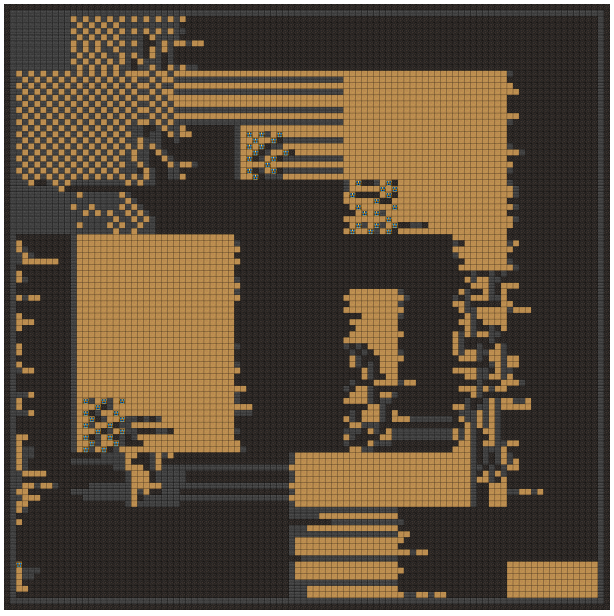

Figura 4.16: Resultado da geração de *Quasi-Hierárquica Markov* do mapa da Figura 4.14. Fonte: Do autor.

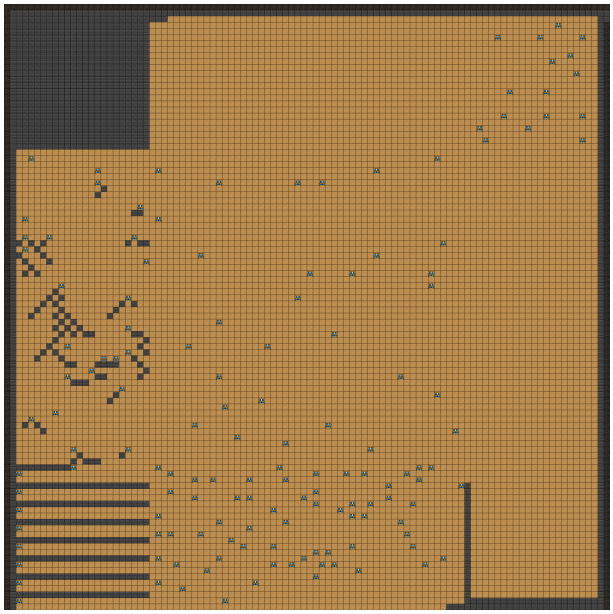

Figura 4.17: Resultado da geração de cadeias de *Markov* hierárquicas do mapa da Figura 4.14. Fonte: Do autor.

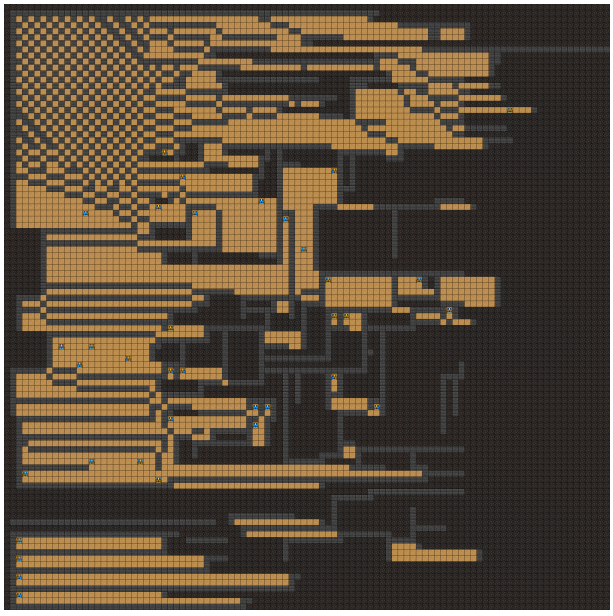

Figura 4.18: Resultado da geração de cadeias de *Markov* hierárquicas do mapa da Figura 4.14 a partir dos estados vazios. Fonte: Do autor.

5 Considerações Finais

Ao longo deste trabalho foi realizada uma pesquisa exploratória sobre a geração procedural de mapas para jogos eletrônicos, utilizando cadeias de *Markov* como método central. Como resultado foi desenvolvido um ambiente online para experimentação e ensino de algoritmos de geração de conteúdo estruturado no MarkovLab. A ferramenta visa suprir uma lacuna educacional ao permitir que desenvolvedores e estudantes testem, comparem e refinem técnicas estocásticas em um contexto controlado, sem a necessidade de implementar um jogo completo.

Podemos considerar que os seguintes objetivos foram alcançados: a implementação dos algoritmos de cadeias de *Markov* segundo os preceitos fornecidos pela bibliografia, além disso foi realizada uma avaliação empírica do sistema com diferentes conjuntos de dados de treinamento (mapas-base) e parâmetros de entrada (dimensões, estilos).

Embora o MarkovLab tenha alcançado a maioria dos objetivos planejados não foi possível realizar uma comparação estatística com tabelas de dados: a dificuldade em armazenar e processar dados massivos gerados durante os experimentos impediu análises quantitativas detalhadas, como a validação estatística das probabilidades de transição contra modelos teóricos.

Acreditamos que o MarkovLab contribui inicialmente para o ensino de técnicas de geração procedural ao esclarecer o uso de cadeias de *Markov*, frequentemente abordadas de forma teórica ou fragmentada em protótipos específicos. Ao centralizar múltiplas abordagens em uma única ferramenta, o projeto facilita a comparação entre métodos e incentiva a experimentação criativa.

A principal limitação do MarkovLab foi a dificuldade de implementar uma identificação automática de padrões. A ausência de um algoritmo para classificação automática de tiles (como cantos, corredores ou salas) limitou a complexidade dos mapas gerados, restringindo a aplicação a cenários mais simplificados.

Como perspectivas para trabalhos futuros, propõe-se a incorporação de temas como:

- `Expansão de Gêneros`: Adaptar o sistema para outros estilos de jogos (ex: plataforma 2D, RPG tático) mediante novos conjuntos de regras e dados de treinamento.

- `Comparação estatística`: implementar métodos e métricas que permitam a realização de análises quantitativas, comparando os resultados dos diferentes métodos de geração procedural.

Bibliografia

BLASKOWSKI, P. Procedural generation via machine learning. 2016.

BLIZZARD. *Diablo*. 1996. [PC CD-ROM].

CAILLOIS, R. *Man Play and Games*. [S.l.]: University of Illinois Press, 1958.

COSTA, G. M.; BORCHARTT, T. B. Procedural terrain generator for platform games using markov chain. 2018.

GEARBOX SOFTWARE. *Borderlands*. 2009. [PC, XBOX 360, PS3].

GREY, D. Procedural generation in game design. Taylor and Francis Group, LLC, p. 3–10, 2017.

HEADUP GAMES. *The binding of isaac*. 2011. [PC].

HELLO GAMES. *No Man's Sky*. 2016. [PC, XBOX ONE, PS4].

HENRY; JEREMY; WORRANAT; ET. *Markov Chains*. 2022. Disponível em: ⟨https://brilliant.org/wiki/markov-chains⟩.

HUIZINGA, J. *Homo Ludens*. 4. ed. São Paulo, SP, Brasil: Editora Perspectiva S/A, 2000.

KLEI ENTERTAINMENT. *Don't Starve*. 2013. [PC, XBOX ONE, PS4, NINTENDO SWITCH].

LU, S.; FOOMANI, A.; LU, S.; CHEN, X. Procedural content generation of the legend of zelda using markov models. 2019.

MOJANG STUDIOS. *Minecraft*. 2009. [PC].

NEWZOO. *Newzoo*. 2024. Disponível em: ⟨https://newzoo.com/resources/trend-reports/newzoos-global-games-market-report-2024-free-version⟩.

NINTENDO. *Super Mario Bros*. 1985. [NES, arcade].

NINTENDO. *The Legend of Zelda*. 1986. [NES,Super Nintendo EntertainmentSystem,Game Boy,Nintendo 64,Game Boy Color,Game Boy Advance,GameCube,Wii,Nintendo DS,Nintendo 3DS,Wii U,Nintendo Switch].

PEREZ-LIEBANA, D.; LIU, J.; KHALIFA, A.; GAINA, R. D.; TOGELIUS, J.; LUCAS, S. M. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. 2019.

SALAMAT, S.; KHALEGHI, B.; IMANI, M.; ROSING, T. Workload-aware opportunistic energy efficiency in multi-fpga platforms. 2019.

SCHONS, L. *PRNG's — Meus números aleatórios são realmente aleatórios?* 2023. Disponível em: ⟨https://medium.com/@sschonss/prngs-meus-n%C3%BAmeros-aleat%C3%B3rios-s%C3%A3o-realmente-aleat%C3%B3rios-7f9cc177bca1⟩.

SHAKER, N.; LIAPIS, A.; TOGELIUS, J.; LOPES, R.; BIDARRA, R. Constructive generation methods for dungeons and levels. In: *Procedural Content Generation in Games*. [S.l.]: Springer, 2016. p. 31–55.

SHAKER, N.; LIAPIS, A.; TOGELIUS, J.; LOPES, R.; BIDARRA, R. *Procedural Content Generation in Games*. [S.l.]: Springer, 2016. 31-54 p.

SHARP, J.; MACKLIN, C. *Games, Design and Play A Detailed Approach to Iterative Game Design*. Brooklyn, New York, EUA: Addison-Wesley, 2016.

SMITH, G. The future of procedural content generation in games. Northeastern University, Playable Innovative Technologies Group, 2014.

SNODGRASS, S. Markov models for procedural content generation. 2018.

SNODGRASS, S.; ONTANóN, S. Experiments in map generation using markov chains. 2014.

SNODGRASS, S.; ONTANóN, S. A hierarchical approach to generating maps using markov chains. 2014.

SNODGRASS, S.; ONTANóN, S. Controllable procedural content generation via constrained multi-dimensional markov chain sampling. 2016.

SNODGRASS, S.; ONTANóN, S. Learning to generate video game maps using markov models. 2016.

SUMMERVILLE, A.; SNODGRASS, S.; GUZDIAL, M.; HOLMGåRD, C.; HOOVER, A. K.; ISAKSEN, A.; NEALEN, A.; TOGELIUS, J. Procedural content generation via machine learning(pcgml). 2018.

TOGELIUS, J.; SHAKER, N.; NELSON, M. J. *Computational Synthesis and Creative Systems*. [S.l.]: Springer International Publishing Switzerland 2016, 2016. 1–15 p.

ZAFAR, A.; IRFAN, A.; SABIR, M. Z. Generating general levels using markov chains. 2019.