Universidade de São Paulo
Escola Politécnica
Programa de Educação Continuada em Engenharia
Especialização em Inteligência Artificial

Matheus Ribeiro de Almeida Veneziani

Explorando geração de conteúdo procedural via aprendizado de máquina: geração de estágios com modelo de difusão

São Paulo
2024

MATHEUS RIBEIRO DE ALMEIDA VENEZIANI

Explorando geração de conteúdo procedural via aprendizado de máquina: geração de estágios com modelo de difusão

— Versão Original —

Monografia apresentada ao Programa de Educação Continuada em Engenharia da Escola Politécnica da Universidade de São Paulo como parte dos requisitos para conclusão do curso de Especialização em Inteligência Artificial.

Orientador: Profa. Dra. Larissa Driemeier

São Paulo
2024

Catalogação-na-publicação

Veneziani, Matheus Ribeiro de Almeida
Explorando geração de conteúdo procedural via aprendizado de máquina: geração de estágios com modelo de difusão/ M.Veneziani – São Paulo, 2024.
19p.

Monografia (Especialização em Inteligência Artificial) – Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia.

1. Geração De Conteúdo 2. Aprendizado De Máquina 3. Modelo De Difusão 4. Videogames.
I. Universidade de São Paulo. Escola Politécnica. PECE – Programa de Educação Continuada em Engenharia. II.t.

Sumário

Resumo

VENEZIANI, M.*Explorando geração de conteúdo procedural via aprendizado de máquina: geração de estágios com modelo de difusão.* 2024. Monografia (Especialização em Inteligência Artificial) – Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia. Universidade de São Paulo, São Paulo, 2024.

Neste estudo, investigamos a geração de estágios de *Super Mario Bros* por meio da Geração de Conteúdo Procedural (PCG) utilizando Aprendizado de Máquina, com foco em um modelo de difusão multinomial baseado na arquitetura *UNet*. Demonstramos que o modelo pode criar estágios viáveis para serem completados por um agente artificial, embora enfrente desafios relacionados ao tempo de geração e ao tamanho dos estágios produzidos. Propomos melhorias para o modelo, especialmente quando consideramos sua utilização em cocriação com *designers*. Este estudo oferece uma introdução ao campo da PCG para jogos, destacando seu potencial em diferentes títulos que empregam representações categóricas, não se limitando apenas a estágios de jogo.

Palavras-chave: Geração De Conteúdo. Aprendizado De Máquina. Modelo De Difusão. Videogames.

Abstract

VENEZIANI, M.*Exploring procedural content generation via machine learning: stage generation with diffusion model.* 2024. Monografia (Especialização em Inteligência Artificial) – Escola Politécnica da Universidade de São Paulo. PECE – Programa de Educação Continuada em Engenharia. University of São Paulo, São Paulo, Brazil. 2024.

In this study, we explore the generation of Super Mario Bros stages through Procedural Content Generation (PCG) using Machine Learning, focusing on a multinomial diffusion model based on the UNet architecture. We demonstrate that the model can create stages viable for completion by an artificial agent, albeit facing challenges related to generation time and the size of produced stages. We propose improvements to the model, particularly when considering its use in co-creation with designers. This study provides an introduction to the field of PCG for games, highlighting its potential across various titles employing categorical representations, not limited solely to game stages.

Keywords: Content Generation. Machine Learning. Diffusion Model. Games.

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A Geração de Conteúdo Procedural (PCG, do inglês *Procedural Content Generation*) envolve a criação algorítmica de conteúdo para jogos, muitas vezes com pouca ou nenhuma interação do usuário. Ela engloba aplicações que geram conteúdo de forma autônoma ou em colaboração com jogadores e/ou desenvolvedores humanos. Dentro desse contexto, o conteúdo abrange uma ampla gama de elementos de jogo, incluindo estágios, mapas, regras, texturas, itens, músicas, personagens e muito mais (Shaker, Togelius e Nelson 2016).

Neste trabalho trataremos principalmente de videogames, mas PCG também pode ser aplicada para outras categorias de jogos (jogos de tabuleiro, quebra-cabeças, cartas, etc.) como mencionado em Shaker, Togelius e Nelson 2016.

PCG é cada vez mais proeminente tanto no desenvolvimento como nas pesquisas relacionadas a jogos. É utilizado para aumentar o valor de repetição de videogames, diminuir esforço e custo de produção, diminuir consumo de espaço de armazenamento, ou simplesmente como uma forma estética. Pesquisas acadêmicas endereçam estes desafios, também investigam como PCG pode oferecer novas formas de experiência, como jogos capazes de se adaptar dinamicamente ao jogador (A. Summerville et al. 2018).

PCG via *Machine Learning* (PCGML) representa os métodos de PCG que utilizam modelos treinados com base em conteúdos de jogos existentes para gerar novos conteúdos diretamente como suas saídas. PCGML compartilha muitas tarefas com outras formas de PCG, como geração autônoma, cocriação e compressão de dados. No entanto, devido ao seu treinamento em artefatos existentes, esses modelos podem ser aplicados de forma mais ampla, incluindo tarefas como correção e análise crítica de novos conteúdos (A. Summerville et al. 2018).

Um dos aspectos cruciais no desenvolvimento de jogos é a criação de estágios, pois representam o espaço virtual dentro do qual a maior parte da interação com os jogadores acontece. Assim sendo, estágios representam um alvo bem atrativo para PCG (A. J.

Summerville et al. 2016). No ramo de pesquisa em PCG, um videogame frequentemente utilizado como base é *Super Mario Bros* (SMB), um jogo cultural e historicamente significante, cujos estágios são baseados em mecânicas de plataforma e compostos por *tiles* (blocos) (Lee e Simo-Serra 2023).

O objetivo deste trabalho é a construção de um modelo para PCGML capaz de gerar novos estágios de SMB, usando como base e inspiração modelos apresentados por Lee e Simo-Serra 2023 e Hoogeboom et al. 2021.

Capítulo 2

Revisão da literatura

PCG é um campo de pesquisa ativo em inteligência artificial. Exemplos de tipos de arquitetura de modelos considerados em pesquisas para geração de recursos são Redes Neurais Convolucionais (CNNs, do inglês *Convolutional Neural Networks*) (Chen et al. 2020), Redes Adversárias Generativas (GANs, do inglês *Generative Adversarial Networks*) (Migdał, Olechno e Podgórski 2021), Autoencoders Variacionais (VAEs, do inglês *Variational Autoencoders*) (Sarkar e Cooper 2021), Modelos de Linguagem de Grande Escala (LLMs, do inglês *Large Language Models*) (Sudhakaran et al. 2023), modelos de difusão (Lee e Simo-Serra 2023) e mais.

Migdał, Olechno e Podgórski 2021 apresentam uma visão geral de métodos utilizados durante o desenvolvimento de um videogame. Os autores abordam algumas tarefas relacionadas a imagens e apresentam exemplos de modelos adequados para cada uma: geração de imagens a partir de um conjunto existente — GAN; super-resolução ou *upscaling* — ESRGAN (do inglês *Enhanced Super-Resolution* GAN); transferência de estilo — GauGAN (GANs para geração de Arte); tradução de imagens e segmentação (tanto supervisionada como não supervisionada) — *UNet*, Tile2Vec.

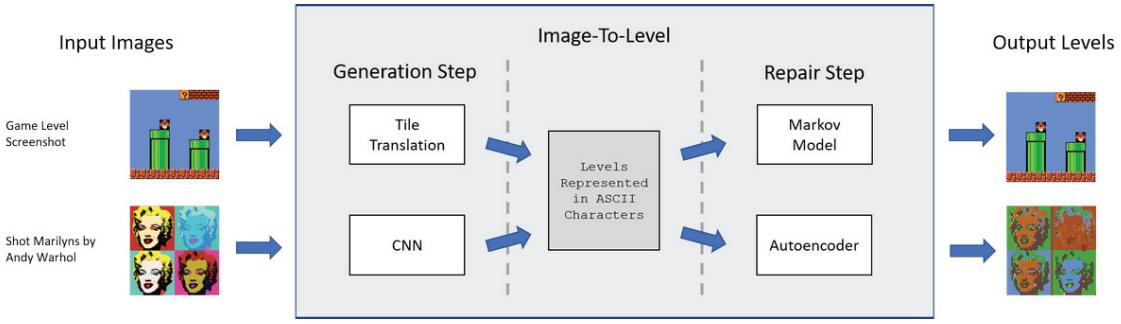
Neste trabalho daremos um foco maior na tarefa de geração de estágios, portanto, apresentaremos uma revisão de alguns trabalhos relacionados.

Chen et al. 2020 propõem um sistema cocriativo que traduz uma dada imagem de entrada (como um esboço feito por um *designer*, por exemplo) num estágio de SMB ou *Lode Runner*. A proposta da ferramenta é proporcionar uma interação mais intuitiva com um modelo de PCGML, dispensando a necessidade de conhecimento profundo por parte do usuário para obter resultados eficazes.

O fluxo de processamento do sistema proposto é composto por duas etapas sequenciais, uma de geração e outra de reparo, como exemplificado na figura 2.1. Na figura, é possível observar que em cada etapa há duas alternativas de processamento. Os autores optaram por esse formato para uma validação mais precisa do fluxo de processamento do sistema,

em vez de focar nas técnicas específicas empregadas em cada etapa. A etapa de geração é a responsável por gerar uma representação em ladrilhos[1] a partir da imagem original, seja por meio de Tradução de Ladrilhos (ou *Tile Translation* em inglês) ou por uma CNN. A saída resultante da geração passa pela etapa de reparo, que trata os blocos gerados e os rearranja de forma que a saída final tenha mais semelhança ao jogo original. Para isso o sistema proposto utiliza um *Autoencoder* ou uma Cadeia de Markov. Os resultados mostram que a execução do processo de geração e reparo tem resultados positivos na criação de estágios baseados em imagens como entrada (Chen et al. 2020).

Figura 2.1: Diagrama do Sistema *Image to Level*



Fonte: extraído de Chen et al. 2020.

Merino et al. 2023 apresentam o *Five-Dollar Model*, um modelo *text-to-image* leve capaz de gerar imagens ou *tile maps* dada uma codificação de *prompt* textual. Não foi construído para SMB, mas testaram o desempenho do modelo na geração de mapas baseados em ladrilhos.

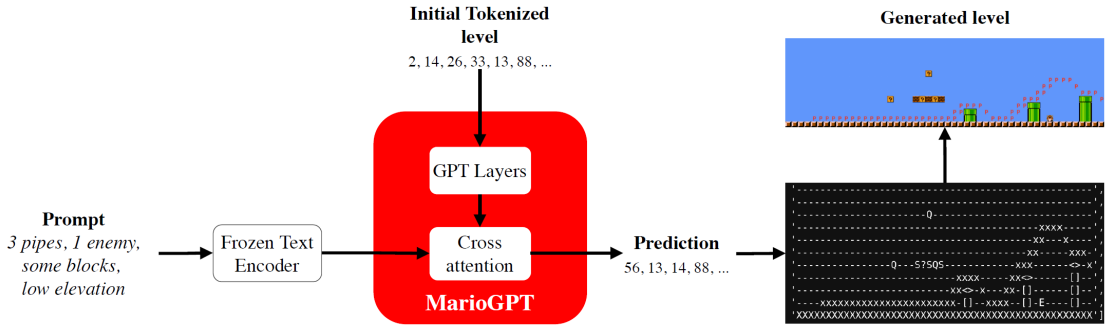
O modelo desenvolvido pelos autores é uma rede *feedforward* relativamente simples, aprimorada com a capacidade de mapear um vetor de *embedding* de sentença concatenado com um ruído randômico em uma representação visual categórica baseada em ladrilhos. Essa transformação é realizada por meio de blocos residuais de camadas convolucionais. A codificação das sentenças de entrada é feita por um modelo *transformer* externo pré-treinado e, por conta da necessidade de aumentação dos dados de entrada (gerar mais sentenças para descrever as imagens de treinamento), ainda utilizaram o GPT-4 para aumentar o conjunto de dados de treinamento do modelo (Merino et al. 2023).

A abordagem apresentada é notável por utilizar modelos pré-treinados para lidar com o desafio inicial da representação das sentenças, que influenciam a saída do modelo. Além disso, a técnica de aumento de dados empregada também é interessante. Surpreendentemente, o gerador empregado não precisa ser excessivamente complexo para alcançar resultados tão impressionantes, como destacado por Merino et al. 2023.

[1] Em computação gráfica e *design* de jogos, ladrilhos ou "*tiles*" são pequenas imagens ou blocos gráficos usados para construir cenários ou ambientes maiores. São tipicamente quadrados e representam elementos individuais, como pedaços de terreno, paredes, objetos, ou qualquer outro componente do ambiente de um jogo.

Sudhakaran et al. 2023 demonstram como LLMs podem ser aplicados para geração de estágios. Os autores apresentam o *MarioGPT*, um modelo baseado em uma versão mais leve do GPT2 chamada *DistilGPT2*. O modelo foi construído para gerar *strings* de *tokens* que representam os ladrilhos de um estágio de SMB.

Figura 2.2: Diagrama do Sistema *MarioGPT*



Fonte: extraído de Sudhakaran et al. 2023.

O *MarioGPT* codifica os *prompts* de entrada utilizando o BART (Lewis et al. 2019) e incorpora a média dos estados escondidos nos pesos de *Cross Attention* das camadas de atenção do GPT2, conforme a figura 2.2. No total, o modelo conta com 96 milhões de parâmetros (Sudhakaran et al. 2023).

Os resultados evidenciam a capacidade do modelo em produzir estágios diversos e em seguir diretrizes fornecidas por um *prompt* textual. Um dos benefícios destacados pelos autores é a capacidade de reutilizar uma arquitetura de LLM, o que permite aproveitar os avanços e melhorias contínuas nesta área (Sudhakaran et al. 2023).

Já Lee e Simo-Serra 2023 propõem uma solução que aplica um modelo de difusão incondicional para gerar estágios de SMB. O sistema é uma adaptação de um modelo de difusão incondicional baseado em uma *UNet*, contando com mecanismos de auto-atenção e *embeddings* temporais.

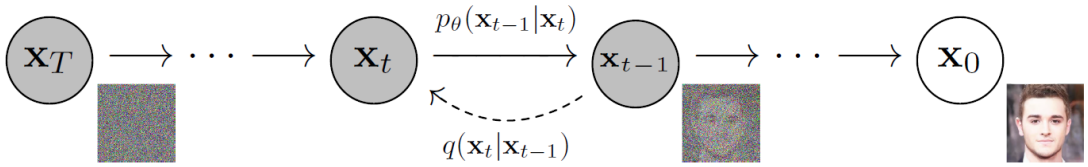
A arquitetura e treinamento do modelo foram adaptados para geração dos estágios de maneira categórica, diferentemente de *Denoising Diffusion Probabilistic Models* (DDPMs) tradicionais que trabalham com imagens e são construídos em torno da representação contínua dos dados. Dentre os ajustes realizados no procedimento, os autores utilizaram uma função de custo customizada (*Reconstruction Loss*) que, por conta da representação categórica, funciona como se fosse uma função de custo *multi-class cross-entropy* (Lee e Simo-Serra 2023).

Além disso, os autores também experimentaram: diferentes esquemas de *Beta scheduling* (especificamente o linear, quadrático e sigmoide) que regulam o nível de ruído inserido em cada passo do processo de difusão; e também com uma escala de temperatura

por *sprite*, pois o parâmetro de temperatura global normalmente utilizado em modelos de difusão (controla o nível de randomização no processo de geração) poderia levar a uma sub ou super-representação de certos *sprites* (Lee e Simo-Serra 2023).

Com base neste breve resumo sobre alguns trabalhos em PCGML, a proposta deste projeto é realizar a construção de um gerador de estágios de SMB utilizando um DDPM. O DDPM é uma cadeia de Markov parametrizada, treinada através de inferência variacional, com o objetivo de produzir exemplos que se assemelhem aos dados de entrada após um período de tempo finito. As transições desta cadeia são habilmente aprendidas para reverter um processo de difusão. Trata-se de uma cadeia de Markov que gradualmente adiciona ruído aos dados de entrada até "desconstruí-los", uma técnica que permite ao modelo capturar as nuances e complexidades dos dados originais de forma eficiente (Ho, Jain e Abbeel 2020). Este processo inverso de difusão é essencial para o DDPM reconstruir com precisão os padrões presentes nos estágios de SMB, resultando em uma geração de conteúdo mais autêntica. A figura 2.3 exemplifica este processo, da direita para a esquerda temos o processo de difusão, que adiciona ruído aos dados a cada passo (tempo), e no sentido contrário ocorre a reversão da difusão.

Figura 2.3: Exemplo de um processo de difusão aplicado a uma imagem.



Fonte: extraído de Ho, Jain e Abbeel 2020.

Capítulo 3

Materiais e Métodos

Neste capítulo vamos apresentar as especificações do conjunto de dados, do modelo gerador e do ambiente utilizado no treinamento.

3.1 Materiais

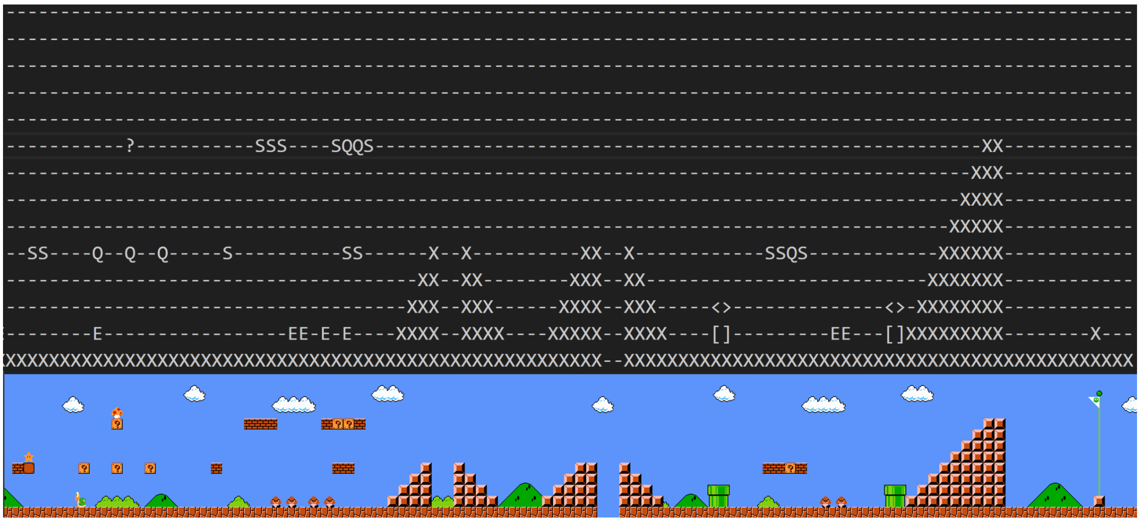
3.1.1 *Dataset*

Os estágios usados durante o treinamento provem do *Video Game Level Corpus* (VGLC) (A. J. Summerville et al. 2016), um conjunto de dados disponível em `https://github.com/TheVGLC/TheVGLC.git`, que contém estágios de diversos jogos anotados em arquivos de texto para facilitar pesquisas em PCG. Neste trabalho utilizamos apenas os arquivos referentes aos estágios processados de "*Super Mario Bros*" e "*Super Mario Bros* 2 *(Japan)*".

Os mesmos estão codificados em texto com cada ladrilho do estágio mapeado a um *token* correspondente como exemplificado na Figura 3.1. Ao todo temos 37 arquivos com representações de estágios e 13 ladrilhos distintos nesse subconjunto da base de dados. Na Tabela 3.1 apresentamos o mapeamento entre os ladrilhos e tokens utilizado na anotação dos arquivos. A Figura 3.2 contém as alguns trechos de vários estágios do *dataset* para termos uma noção da variação entre eles.

Para reproduzir os estágios em forma gráfica e também para verificar se são ao menos jogáveis (se podem ser atravessados de uma ponta a outra por um agente) utilizamos o *Mario AI Framework* (Ahmed 2022) apresentado inicialmente por Karakovskiy e Togelius 2012. O uso previsto do *framework* é exclusivamente para pesquisa e compreende ferramentas destinadas a testar os estágios gerados na notação do VGLC, seja por meio de agentes autônomos ou mesmo pela interação direta do usuário. Isso possibilita uma avaliação abrangente e detalhada dos estágios produzidos, permitindo que pesquisadores

Figura 3.1: Trecho de estágio de *Super Mario Bros* disponível no VGLC. Acima temos o arquivo processado e abaixo o original.



Fonte: Autoria própria com composição de estágio e anotação extraídos do VGLC (A. J. Summerville et al. 2016).

Tabela 3.1: Mapeamento entre *tokens* e ladrilhos utilizado no VGLC com suas respectivas contagens no conjunto de dados.

Token	**Descrição**	**Ladrilhos**	**Quantidade**
‘X’	Ladrilhos indestrutíveis no geral		9.542
‘S’	Ladrilhos destrutíveis		2.339
‘-’	Vazio		91.398
‘?’ e ‘Q’	Caixas com ou sem itens		339
‘E’	Inimigos no geral		614
‘<’	Topo esquerdo dos canos		198
‘>’	Topo direito dos canos		197
‘[’	“Corpo” esquerdo dos canos		477
‘]’	“Corpo” direito dos canos		479
‘o’	Moedas		459
‘B’	Topo dos canhões		39
‘b’	“Corpo” dos canhões		25

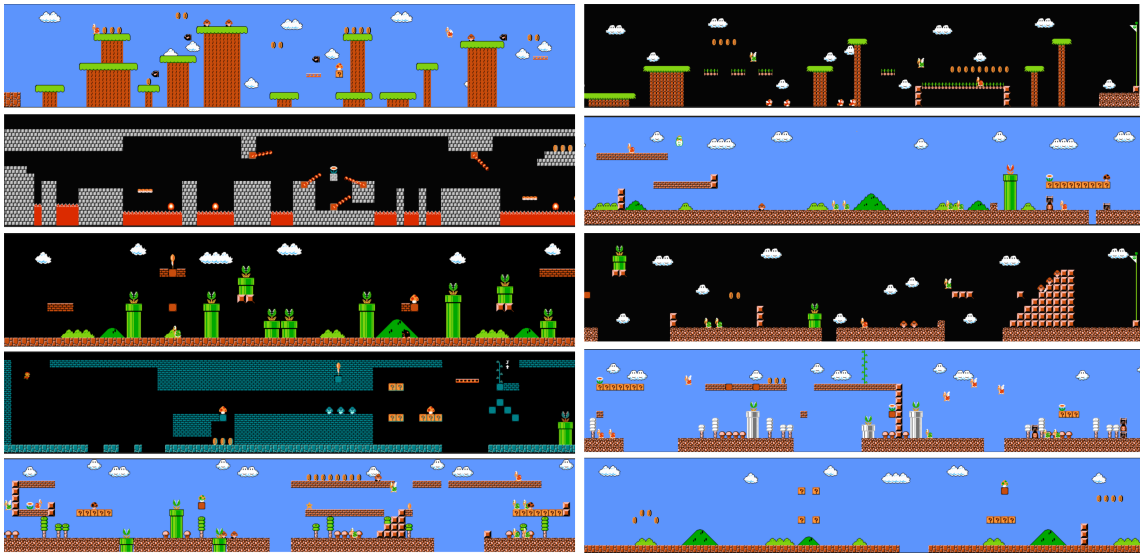
Fonte: Autoria própria com as imagens provindas do *Mario AI Framework* (Ahmed 2022) e a partir do conjunto de dados anotado do VGLC (A. J. Summerville et al. 2016).

e desenvolvedores entendam melhor a qualidade e a jogabilidade dos níveis gerados por seus modelos.

3.1.2 Tratamento dos dados

O processo de tratamento dos dados foi orientado pelas diretrizes estabelecidas por Lee e Simo-Serra 2023, uma das referências fundamentais na concepção do modelo. Devido à utilização de estágios de dois títulos distintos, é natural que ocorram disparidades nas

Figura 3.2: Trechos de estágios originais que estão anotados no VGLC.



Fonte: Autoria própria com base em trechos de imagens extraídas do VGLC (A. J. Summerville et al. 2016).

anotações, especialmente em relação à altura deles. Para garantir a consistência das representações e montar um conjunto de treinamento coeso, foram realizadas algumas adaptações para uniformizar as características dos estágios.

Primeiramente reduzimos os blocos considerados de 13 para 11, isso porque dois deles aparecem com pouca frequência no conjunto de dados, como pode ser visto na Tabela 3.1. Os *tokens* substituídos por '-' foram 'b' e 'B', que representam blocos de torre e canhão respectivamente.

O segundo passo foi uniformizar as alturas dos estágios de *Super Mario Bros* 2 para ficarem consistentes com os 14 blocos de altura em *Super Mario Bros.* Para tanto realizamos operações para adicionar ou remover linhas do início, ou do final dos estágios que apresentam alturas diferentes de 14, conforme descrito na Tabela 3.2.

Tabela 3.2: Estágios e operações aplicadas para uniformizar as alturas. A representação numérica do estágio significa o arquivo na posição correspondente na lista de arquivos processados de *Super Mario Bros* 2 *(Japan)* ordenados de forma crescente pelo nome.

Estágio	Operação
1	Duplicar primeira e última linha
7	Remover primeira e última linha
15	Duplicar ultima linha
18	Remover ultima linha
3, 4, 8, 9, 13, 14, 16, 20, 21 e 22	Remover primeira linha

Fonte: Autoria própria.

Feitos os ajustes necessários, para gerar o conjunto de treinamento para o modelo extraímos de cada estágio um conjunto de janelas de dimensão 14x14 blocos utilizando uma janela deslizante de 1 bloco, aproveitando para substituir os *tokens* textuais por

números que representam as classes de cada bloco seguindo a Tabela 3.3. Ao final do processamento, o conjunto de dados é categórico e contempla 6.961 janelas 14x14.

Tabela 3.3: Mapeamento dos *tokens* em inteiros.

Token	Valor
'X'	0
'S'	1
'-'	2
'?'	3
'Q'	4
'E'	5
'<'	6
'>'	7
'['	8
']'	9
'o'	10

Fonte: Autoria própria.

3.1.3 Instrumentos

Para executar as análises e o treinamento do modelo, utilizamos um computador com as seguintes especificações:

- Sistema Operacional: WSL2 Ubuntu (rodando em *Windows* 11)
- Python: 3.10.12
- Tensorflow: 2.15
- Placa de vídeo: NVIDIA GeForce GTX 1050 Ti

3.2 Métodos

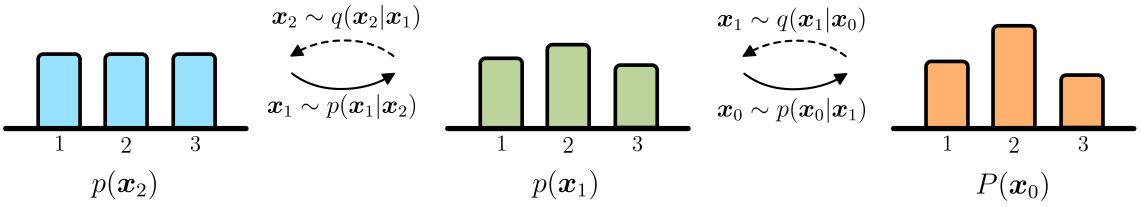
3.2.1 Definição do modelo

Inicialmente, tentamos construir um modelo semelhante ao apresentado por Lee e Simo-Serra 2023, porém os resultados não foram satisfatórios devido à falta de informação para reproduzir o modelo conforme descrito. Diante dessa situação, buscamos outras referências e encontramos o Modelo de Difusão Multinomial (Hoogeboom et al. 2021), que se mostrou uma cujo código-fonte está disponível em: https://github.com/ehoogeboom/multinomial_diffusion.git. Essa mudança de abordagem foi fundamental para avançarmos em nossa pesquisa e alcançarmos resultados mais significativos.

Hoogeboom et al. 2021 descrevem o *framework* do Modelo de Difusão Multinomial que funciona como um DDPM convencional, mas com a diferença que define o processo de difusão com base na distribuição categorica dos dados. Sendo assim, ao invés de trabalhar com os ruídos diretamente na imagem, a difusão acontece na distribuição das categorias adicionando um ruído uniforme a cada unidade de tempo.

A Figura 3.3 ilustra esse processo. Inicialmente, temos uma distribuição inicial $P(x_0)$. Em seguida, aplicamos um processo de difusão $q(x_t|x_{t-1})$ que adiciona ruído uniforme a cada passo resultando em $p(x_2)$. Por fim, aplicamos o processo reverso $p(x_{t-1}|x_t)$, que remove o ruído, nos levando de volta à distribuição original. Esse ciclo de difusão e reversão é fundamental para o funcionamento do modelo de difusão probabilística.

Figura 3.3: Exemplo do funcionamento do Modelo de Difusão Multinomial.



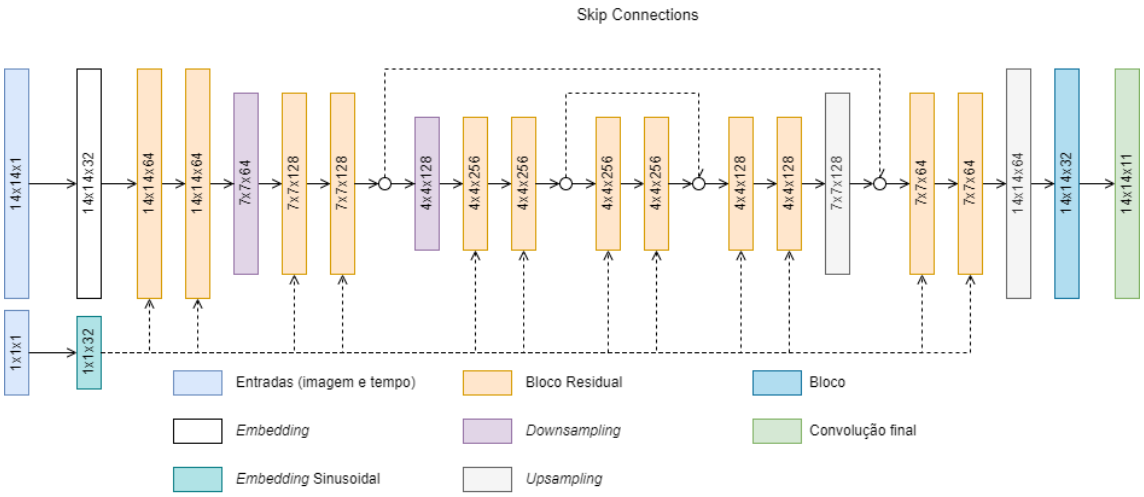
Fonte: Extraído de Hoogeboom et al. 2021.

Adaptamos uma parte do código-fonte relacionada ao modelo de difusão para o ambiente do *TensorFlow* e configuramos a arquitetura do modelo de acordo com o diagrama mostrado na Figura 3.4. Essa adaptação foi essencial para integrar o modelo de difusão ao nosso ambiente de desenvolvimento e garantir sua compatibilidade com as demais ferramentas e bibliotecas utilizadas no projeto. Seguindo os exemplos de Lee e Simo-Serra 2023 e Hoogeboom et al. 2021, utilizamos uma arquitetura similar a *UNet* para o modelo, porém sem as camadas de atenção linear.

Elucidando a arquitetura representada na Figura 3.4, um "Bloco" na arquitetura é composto por uma camada de convolução 2D, seguida por uma camada de *Layer Normalization* e, por fim, uma camada de ativação utilizando a função "gelu". Por sua vez, um "Bloco Residual" consiste em dois "Blocos", uma camada densa e uma camada de ativação "gelu", sendo estes dois últimos utilizados para processar o *Embedding* Sinusoidal que representa o tempo de difusão. No fluxo de processamento, a entrada inicial é primeiro submetida a um "Bloco", e o resultado é então adicionado ao tempo de processamento. Em seguida, o resultado passa pelo segundo "Bloco". Finalmente, a entrada original é adicionada ao resultado, preservando assim a conexão residual.

Aplicamos a função de ativação "gelu" e o tamanho de kernel 3 nas convoluções 2D ao longo da rede, exceto na saída que é o resultado de uma camada de convolução com 11 filtros, kernel 1 e ativação linear. Nas camadas de *DownSampling* e *UpSampling*, optamos

Figura 3.4: Diagrama da arquitetura do modelo utilizado neste trabalho.



Fonte: Autoria própria.

por utilizar *MaxPooling* e *UpSampling* com interpolação bilinear. No entanto, tivemos que proceder com cautela devido às dimensões originais das janelas de entrada. Por exemplo, durante a transição de uma dimensão de 7x7 para 4x4, empregamos uma camada *MaxPooling* com tamanho de 4 e um stride de 1. Da mesma forma, durante a transição de 4x4 para 7x7, em vez de utilizar uma camada de *UpSampling*, implementamos uma convolução transposta com 7 filtros e um kernel de tamanho 4 para ajustar as dimensões adequadamente. Ao todo modelo contempla $6.844.363$ parâmetros treináveis.

3.2.2 Treinamento

As configurações para o treinamento foram as seguintes:

- Número de passos de difusão: 1000
- Número de épocas: 1000
- Otimizador: *AdamW* com taxa de aprendizado $1e{-}3$ e *weight decay* $1e{-}4$
- Função de custo: *Negative Log Likelihood* em bits por dimensão conforme definição aplicada em Hoogeboom et al. 2021

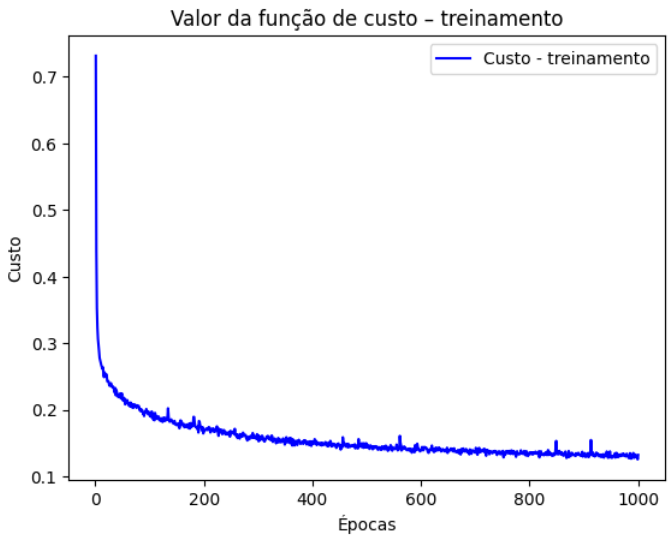
Como o nosso objetivo é construir um gerador e não um classificador, todo o conjunto de dados foi utilizado no treinamento e não houve separação de conjuntos de validação e teste. Os lotes de entrada do treinamento tinham 64 exemplos cada e foram embaralhados.

Capítulo 4

Resultados e Discussão

Durante as 1000 épocas de treinamento a progressão da função de custo do modelo foi de $\sim 0{,}73$ para $\sim 0{,}13$, como pode ser visto na Figura 4.1. Analisando o gráfico, também é possível perceber que a partir da época 500 (onde o custo era $\sim 0{,}14$), o modelo não teve um ganho muito significativo então poderíamos ter encerrado o processo antecipadamente. O tempo total do treinamento do modelo foi cerca de 1 dia, com cada época levando aproximadamente 90 segundos (na máquina com as especificações definidas na Subseção 3.1.3).

Figura 4.1: Valor da função de custo do modelo para cada época de treinamento.

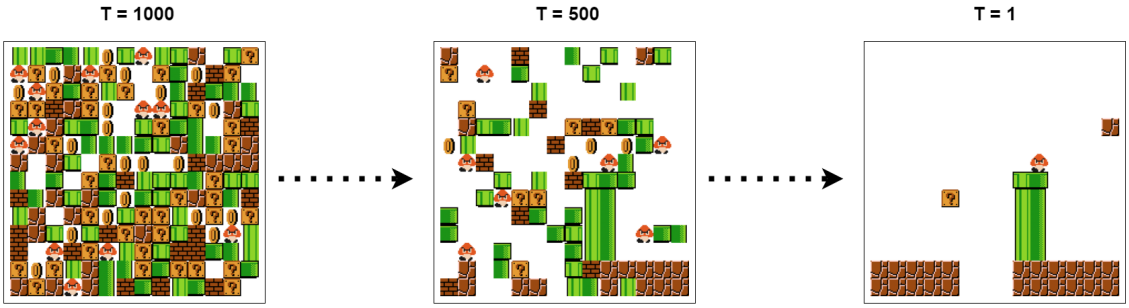


Fonte: Autoria própria.

Com o modelo treinado, passamos para a etapa de geração de estágios. Basicamente, um modelo de difusão baseado no DDPM gera novos dados através da execução repetida do modelo (aqui iteramos 1000 vezes), começando com um ruído uniforme inicial. Por exemplo, a Figura 4.2 ilustra como a entrada evolui ao longo desse processo de reversão

da difusão: no início, é quase puro ruído; na metade, começa a mostrar características da imagem final; e nas iterações finais, temos uma janela 14x14 que poderia ser parte de um estágio de SMB.

Figura 4.2: Exemplo do processo de reversão da difusão utilizado para gerar novos estágios.



Fonte: Autoria própria.

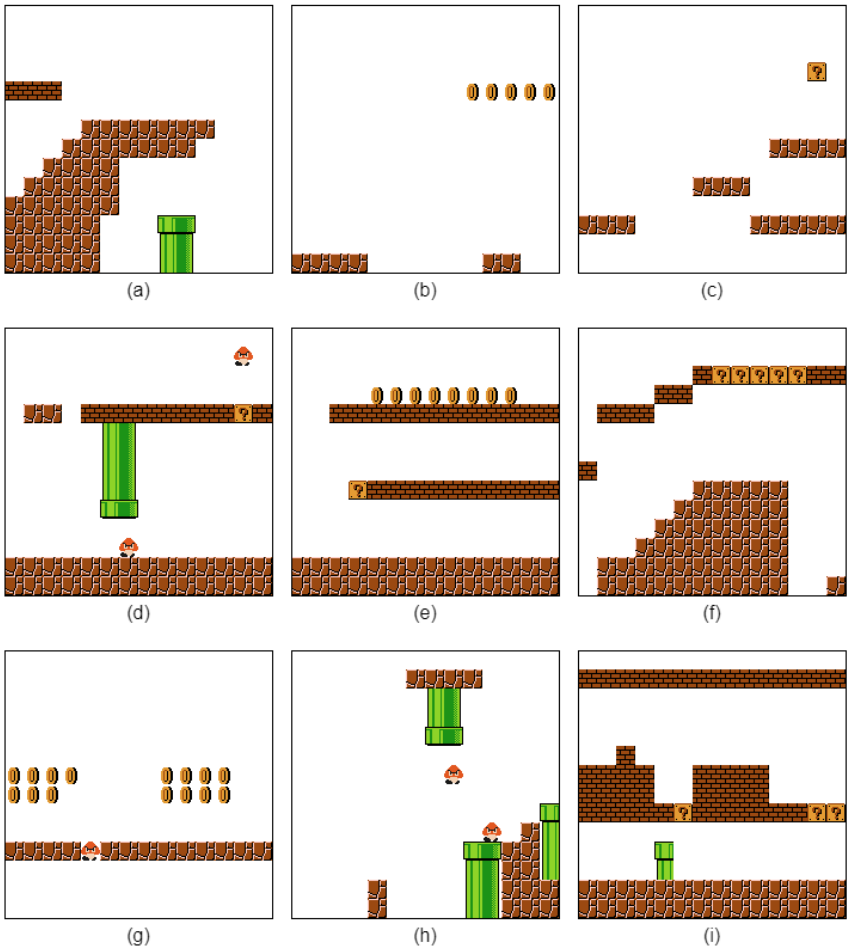
A Figura 4.3 mostra alguns estágios gerados pelo modelo. É possível perceber que a saídas não são perfeitas, com casos em que inimigos são gerados no chão (Figura 4.3(g)), canos gerados pela metade (Figura 4.3(i)) e moedas inacessíveis (Figura 4.3(b)). Porém, ao realizar uma análise qualitativa dos resultados, observamos que o modelo conseguiu capturar alguns padrões dos dados originais e é capaz de gerar novos estágios. No entanto, algumas características evidenciam que o processo de geração ocorreu sem intervenção humana.

Com o objetivo de validar o modelo e comparar com os resultados apresentados por Lee e Simo-Serra 2023, também utilizamos o *Mario AI Framework* (Ahmed 2022) para processar os estágios gerados e avaliar quantos deles são de fato "jogáveis" por um agente A*. Para tanto utilizamos duas abordagens, a primeira considerando somente janelas individuais (14x14) e a outra considerando 12 janelas concatenadas (14x168) para executar o agente.

Para a primeira avaliação consideramos 1000 estágios criados como os da Figura 4.3 e os resultados estão na Tabela 4.1. É possível perceber que, com essa abordagem, a maioria dos estágios gerados pelo modelo é "jogável" (cerca de 91%). Comparando com o que é apresentado por Lee e Simo-Serra 2023, a proporção de estágios jogáveis obtida pelo modelo se aproxima das obtidas pelos modelos apresentados. Isso evidencia que até mesmo um modelo mais simples tem a capacidade de gerar níveis completáveis.

Já para a segunda avaliação realizamos a composição de 1170 estágios maiores, contendo 12 janelas concatenadas. A Tabela 4.2 mostra a proporção de estados finais do agente A* rodando nestes exemplos e os resultados evidenciam que a mera concatenação de saídas costuma gerar estágios impossíveis de terminar. A Figura 4.4 exemplifica uma concatenação que resultou em um estágio com um buraco muito grande sem plataformas,

Figura 4.3: Exemplos de estágios gerados pelo modelo.



Fonte: Autoria própria.

Tabela 4.1: Proporcão de resultados de sucesso do agente A* do *Mario AI Framework*, rodando em 1000 estágios compostos por 1 janela 14x14 comparados com os resultados apresentados por Lee e Simo-Serra 2023.

Modelo	**Jogabilidade**
Modelo Proposto	0,91
Modelo por Lee e Simo-Serra 2023	0,93

Fonte: Autoria própria com base nos resultados da execução do Agente A* do *Mario AI Framework* (Ahmed 2022) e nos resultados apresentados por Lee e Simo-Serra 2023.

impossível de atravessar. Isso destaca uma limitação do modelo na geração de níveis viáveis e desafiadores.

O agente conseguiu resolver somente 20% dos estágios criados deta maneira, um dos exemplos destes está na Figura 4.5. Esse fenômeno acontece por conta das características do modelo, atualmente não há como garantir que as janelas de estágios geradas tenham continuidade entre si. Possivelmente, uma abordagem que explorasse mais detalhada-

mente o espaço latente, utilizando vetores de ruído que estejam "próximos" entre si como entrada, poderia reduzir esse problema. No entanto, essa abordagem implicaria em sacrificar parte da diversidade proporcionada pelo uso de vetores aleatórios como entrada.

Tabela 4.2: Proporcão de estados finais do agente A* do *Mario AI Framework*, rodando em 1170 estágios compostos por 12 janelas 14x14.

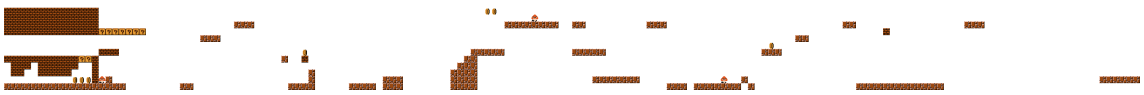
Estado final	Quantidade	Proporção
Venceu	236	$\sim 0{,}20$
Perdeu	849	$\sim 0{,}72$
Timeout	85	$\sim 0{,}08$

Fonte: Autoria própria com base nos resultados da execução do Agente A* do *Mario AI Framework* (Ahmed 2022).

Quando comparamos estes resultados com modelos baseados em LLMs como o MarioGPT que consegue, até certo ponto, gerar estágios compridos seguindo indicações do usuário (Sudhakaran et al. 2023), fica clara uma deficiência do modelo proposto na tarefa de geração de estágios longos.

Entretanto, é crucial considerar tanto o tamanho quanto o funcionamento de ambos os modelos, visto que o MarioGPT possui 96 milhões de parâmetros em comparação com os $\sim$ 7 milhões do modelo proposto, além de operarem de maneiras completamente distintas. Outra alternariva seria expandir o escopo do modelo de difusão ao ampliar a janela de entrada para incluir um número maior de colunas e gerar estágios inteiros de uma vez ao invés de janelas menores.

Figura 4.4: Exemplo de estágio impossível de completar com o agente A*.



Fonte: Autoria própria.

Figura 4.5: Exemplo de estágio possível de completar com o agente A*.

Fonte: Autoria própria.

Em termos do potencial uso do modelo, devido ao tempo necessário para a geração dos estágios, não seria ideal utilizá-lo durante o jogo. Por exemplo, o tempo de processamento para gerar 1000 janelas 14x14 foi de 23 minutos e 42 segundos, enquanto para 1000 estágios com 12 janelas o processo levou aproximadamente 6 horas e 10 minutos. Existe ainda a possibilidade de estágios impossíveis serem criados, frustrando os jogadores.

Para criar estágios mais longos e coerentes a abordagem ideal seria uma de cocriação entre o modelo e um *designer*, similar ao conceito apresentado no "*Image to Level*" por Chen et al. 2020. Dessa forma, seria possível compor janelas geradas de modo que o estágio fosse mais refinado e viável para conclusão, além de resolver inconsistências na geração, como objetos criados de forma incompleta ou mal posicionados no chão.

Capítulo 5

Conclusão

PCG (do inglês, *Procedural Content Generation*) refere-se a algoritmos que podem criar conteúdo de jogo por conta própria, ou junto com jogadores e/ou *designers* humanos. Neste trabalho exploramos uma parte do universo PCGML (do inglês, PCG *via Machine Learning*) para tarefa de geração de estágios em jogos e as diversas técnicas utilizadas em sua execução. Dentro deste escopo, baseando-se em um estudo da literatura, apresentamos um modelo de difusão[1] gerador de estágios de "*Super Mario Bros*" (SMB) capaz de criar níveis previamente inexistentes. Este modelo é uma ferramenta eficaz de cocriação no desenvolvimento de jogos, dadas suas limitações em termos de tempo de geração e qualidade na criação de estágios mais longos. Em termos de escopo deste tipo de modelo, esta arquitetura é capaz de lidar com outros jogos que possúem recursos representados de forma categórica, não só estágios mas também mapas como exemplo.

Para treinar o modelo proposto, utilizamos parte do *dataset* disponível no *Video Game Level Corpus* (VGLC, por A. J. Summerville et al. 2016). O subconjunto utilizado contém arquivos de texto com anotações representando os ladrilhos (ou *tiles*, em inglês) de cada estágio dos jogos "*Super Mario Bros*" e "*Super Mario Bros 2 (Japan)*", para montar o conjunto de treinamento utilizamos janelas categóricas (14x14) extraídas destas anotações. Já na construção do modelo, nos baseamos na proposta do Modelo de Difusão Multinomial apresentado por Hoogeboom et al. 2021 e utilizamos uma arquitetura similar a *UNet*. Por conta destas escolhas o modelo proposto possui as limitações mencionadas anteriormente.

Analisando os estágios gerados mostramos que o modelo aprendeu alguns padrões dos dados originais e é capaz de criar novos níveis, mas com algumas características que evidenciam que o processo de geração ocorreu sem intervenção humana. Já na avaliação de jogabilidade dos estágios resultantes, vemos que o modelo é capaz de gerar janelas 14x14 completáveis em cerca de 91% das vezes. Porém quando realizamos o processo de concatenação para obter níveis mais longos (sem intervenção externa) a porcentagem de

[1] O código-fonte utilizado para o modelo estará disponível neste link.

estágios completáveis cai significativaemente, chegando a aproximadamente 20% para concatenações de 12 janelas. Observamos outra característica importante do modelo que que é o tempo de geração, que torna o modelo mais adequado a cenários de cocriação do que geração em tempo de jogo.

Acreditamos que os resultados foram satisfatórios, mesmo que o modelo seja mais simples que os demais encontrados na literatura. Isso porque não possui diversos elementos encontrados nestes outros como: camadas de atenção, escala de temperatura baseada na probabilidade de cada ladrilho e mais. Porém o modelo não é muito efetivo na construção de estágios maiores sem intervenção externa. Para melhorar isso, poderíamos ampliar a janela de entrada e saída do modelo, para considerar estágios maiores e não somente janelas 14x14. Essa solução não resolve o problema de concatenação de várias saídas do modelo, para tanto devemos trocar o tipo de modelo utilizado ou alterar a arquitetura para gerar um modelo de difusão condicional e talvez assim gerar janelas que podem ser concatendas sem muitos problemas.

Como próximos passos poderíamos pensar em expandir o uso deste modelo para outros jogos ou estudar as possibilidades de alterar o mesmo para que seja mais efetivo na geração de estágios de SMB. Seja pela a inclusão de mecanismos como camadas de atenção e escalas de temperatura no treinamento ou pela alteração na arquitetura de modo a torná-lo condicional ou ampliando as janelas de entrada e saída.

Referências

Ahmed, Khalifa (2022). *Mario-AI-Framework*. https://github.com/amidos2006/Mario-AI-Framework.

Chen, Eugene et al. (out. de 2020). "Image-to-Level: Generation and Repair". Em: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 16.1, pp. 189–195. DOI: 10.1609/aiide.v16i1.7429. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/7429.

Ho, Jonathan, Ajay Jain e Pieter Abbeel (2020). "Denoising Diffusion Probabilistic Models". Em: *Advances in Neural Information Processing Systems*. Ed. por H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 6840–6851. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Hoogeboom, Emiel et al. (2021). *Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions*. arXiv: 2102.05379 [stat.ML].

Karakovskiy, Sergey e Julian Togelius (2012). "The Mario AI Benchmark and Competitions". Em: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1, pp. 55–67. DOI: 10.1109/TCIAIG.2012.2188528.

Lee, Hyeon Joon e Edgar Simo-Serra (2023). "Using Unconditional Diffusion Models in Level Generation for Super Mario Bros". Em: *2023 18th International Conference on Machine Vision and Applications (MVA)*, pp. 1–5. DOI: 10.23919/MVA57639.2023.10215856.

Lewis, Mike et al. (2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. arXiv: 1910.13461 [cs.CL].

Merino, Timothy et al. (out. de 2023). "The Five-Dollar Model: Generating Game Maps and Sprites from Sentence Embeddings". Em: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 19.1, pp. 107–115. DOI: 10.1609/aiide.v19i1.27506. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/27506.

Migdał, Piotr, Bartłomiej Olechno e Błażej Podgórski (2021). *Level generation and style enhancement – deep learning for game development overview*. arXiv: 2107.07397 [cs.CV].

Sarkar, Anurag e Seth Cooper (2021). *Generating and Blending Game Levels via Quality-Diversity in the Latent Space of a Variational Autoencoder*. arXiv: 2102.12463 [cs.LG].

Shaker, Noor, Julian Togelius e Mark Nelson (jan. de 2016). *Procedural Content Generation in Games*. ISBN: 978-3-319-42714-0. DOI: 10.1007/978-3-319-42716-4.

Sudhakaran, Shyam et al. (2023). *MarioGPT: Open-Ended Text2Level Generation through Large Language Models*. arXiv: 2302.05981 [cs.AI].

Summerville, Adam et al. (2018). "Procedural Content Generation via Machine Learning (PCGML)". Em: *IEEE Transactions on Games* 10.3, pp. 257–270. DOI: 10.1109/TG.2018.2846639.

Summerville, Adam James et al. (2016). *The VGLC: The Video Game Level Corpus*. arXiv: 1606.07487 [cs.HC].