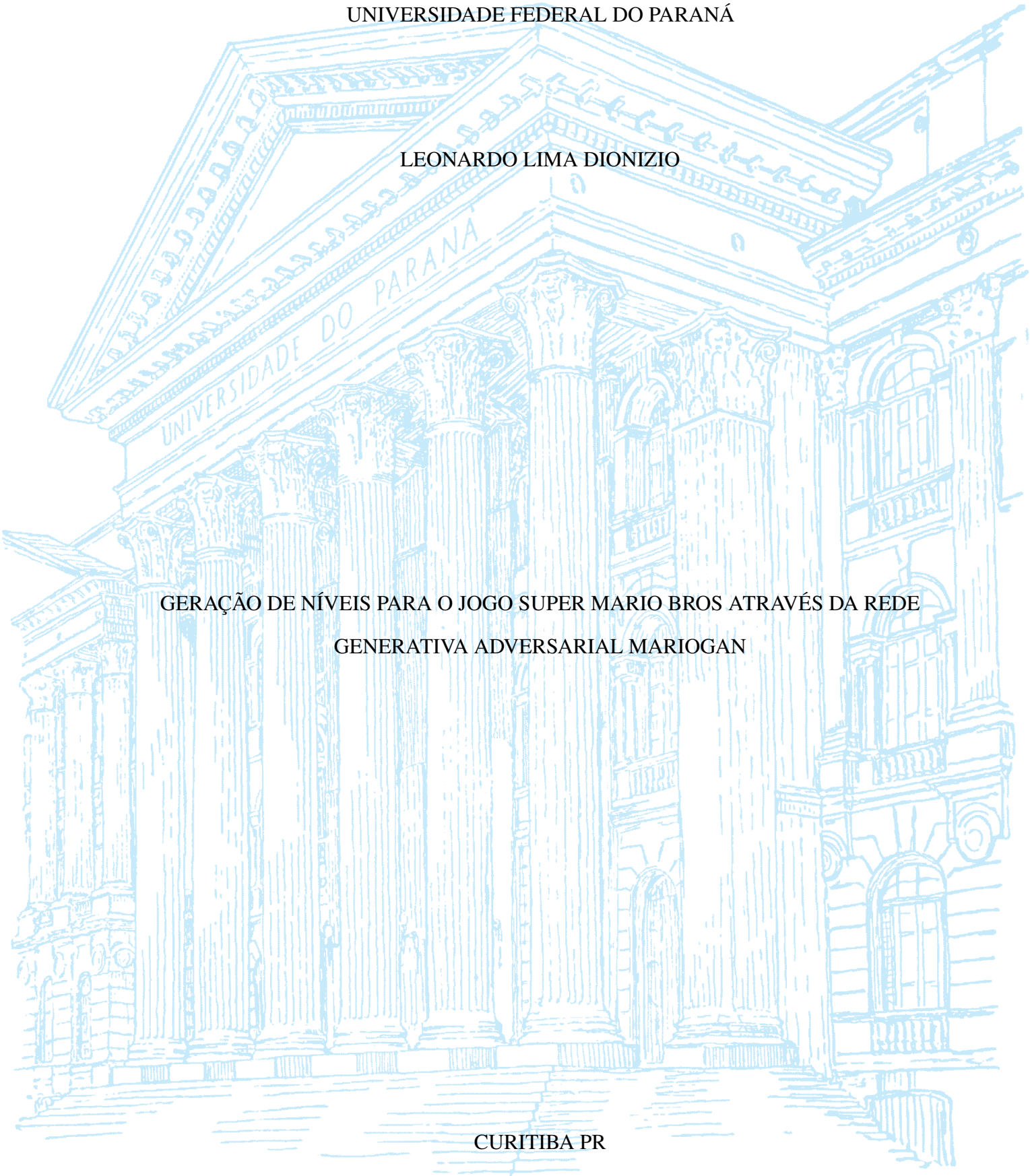
UNIVERSIDADE FEDERAL DO PARANÁ

LEONARDO LIMA DIONIZIO



GERAÇÃO DE NÍVEIS PARA O JOGO SUPER MARIO BROS ATRAVÉS DA REDE GENERATIVA ADVERSARIAL MARIOGAN

CURITIBA PR

2023

LEONARDO LIMA DIONIZIO

GERAÇÃO DE NÍVEIS PARA O JOGO SUPER MARIO BROS ATRAVÉS DA REDE GENERATIVA ADVERSARIAL MARIOGAN

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo J. Spinosa.

CURITIBA PR

2023

AGRADECIMENTOS

À minha mãe, Luciana Camargo Lima Penante, e Talyta Emanuhely Araujo Auffinger, por terem me apoiado nos períodos de dificuldade tanto no curso quanto na realização deste trabalho.

Ao Professor Doutor Eduardo Jaques Spinosa, pela orientação e pelo apoio durante a realização do trabalho.

Aos professores participantes da banca avaliadora, por terem aceitado fazer parte da banca de meu trabalho.

À todos e todas as amizades que fiz no curso e no Centro Acadêmico Alexandre Direne, que compartilharam comigo momentos de alegria e dificuldade nestes cinco anos de graduação.

Ao Grupo CiDAMO, em especial aos Professores Doutores Abel Soares Siqueira, Lucas Garcia Pedroso e Kally Chung, por terem me aceitado no grupo após recusas de múltiplos projetos e me mostrado o caminho para a profissão que desejo seguir.

E, finalmente, à meu falecido pai, Luis Dionizio Júnior, por sempre ter me apoiado nos estudos e no que eu precisava.

RESUMO

A geração de conteúdo procedural é extremamente relevante e desafiadora na indústria dos videojogos, tópico que se tornou ainda mais atual com os modelos neurais profundos. Em especial, o aprendizado adversarial é muito utilizado para a geração de níveis de jogos 2D. Este trabalho utiliza um modelo neural adversarial generativo conhecido para gerar níveis do jogo *Super Mario Bros*(1983). Para diversificar ainda mais os níveis gerados, foram utilizadas técnicas que impõem padrões para o modelo generativo seguir que tem como base algoritmos da computação evolutiva (tanto mono-objetivo quanto multi-objetivo), bem como conjuntos de treinamento diversos. Os algoritmos evolutivos foram comparados em desempenho, tempo de execução, adesão ao padrão do conjunto de treinamento e jogabilidade, avaliação esta última feita por um agente conceituado na literatura. Os resultados permitem gerar níveis diversos e jogáveis, além de manipular estes níveis para atenderem propriedades e dificuldades desejadas.

Palavras-chave: Geração de Conteúdo Procedural. MarioGAN. LVE.

ABSTRACT

Procedural content generation is highly relevant and challenging in the video game industry, a topic that has become even more pertinent with the advent of deep neural models. Particularly, adversarial learning is widely employed for generating 2D game levels. This study utilizes a well-known generative adversarial neural model to create levels for the game Super Mario Bros (1983). To further diversify the generated levels, techniques grounded in evolutionary computing algorithms imposing patterns for the generative model are employed (both single-objective and multi-objective), as well as diverse training sets. Evolutionary algorithms are compared in terms of performance, execution time, adherence to training set patterns, and gameplay, the latter evaluated by an agent well-regarded in the literature. The results demonstrate the ability to generate diverse and playable levels, as well as the capacity to manipulate these levels to meet desired properties and difficulties.

Keywords: Procedural Content Generation. MarioGAN. LVE.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

GAN	*Generative Adversarial Network*
WGAN	*Wassertein Generative Adversarial Network*
PCGML	*Procedural Content Generation using Machine Learning*
LVE	*Latent Variable Evolution*
CMA-ES	*Covariance Matrix Adaptation Evolutionary Strategy*
PSO	*Particle Swarm Optimization*
NSGA	*Non-Dominated Sorting Genetic Algorithm*
MCDMA	*Multiple Criteria Decision Making*
CP	*Compromise Programming*
A*	A Estrela

LISTA DE SÍMBOLOS

$\bar{x}$	Média aritmética
σ	*step size*
λ	População de um algoritmo evolutivo
μ	Número de candidatos sujeitos a cruzamento

SUMÁRIO

1 INTRODUÇÃO

No início da adesão aos modelos neurais, sejam eles profundos ou com pequeno número de camadas neurais, tais modelos se popularizaram inicialmente nas tarefas de regressão e classificação, gerando competições populares como a ImageNet, competição em que o intuito é classificar imagens corretamente dentre 1000 classes. Ainda assim, aproximar uma distribuição, isto é, estimar a máxima verossimilhança é dito possível com maior precisão por (Goodfellow et al., 2014), onde os autores propõem as Redes Neurais Adversariais (GANs). O paradigma do aprendizado adversarial permitiu, através do uso de uma rede geradora e uma rede discriminadora, que uma rede neural profunda se aproxime de uma distribuição de forma a, ao invés de tomar uma decisão em relação a sua entrada, gerar conteúdo que se assemelhe a um conjunto conhecido.

Em (Goodfellow et al., 2014), o intuito desta geração de conteúdo era a geração de imagens. Com o refinamento, este modelo tornou-se popular em outras tarefas, como a transferência de estilo proposta em (Zhu et al., 2020) e a tradução de texto para imagem ilustrada em (Reed et al., 2016). Embora as redes generativas adversariais tenham tornado-se bastante conhecida nestas tarefas relacionadas com imagens, suas aplicações vão muito além destas. Uma das indústrias que muito se beneficia da geração de conteúdo é o ramo dos videojogos. Embora a geração de conteúdo em jogos esteja presente desde o jogo de 1984 *Elite*, GANs são muito utilizadas na geração para jogos, principalmente para níveis, conforme a **Figura 1.1**.

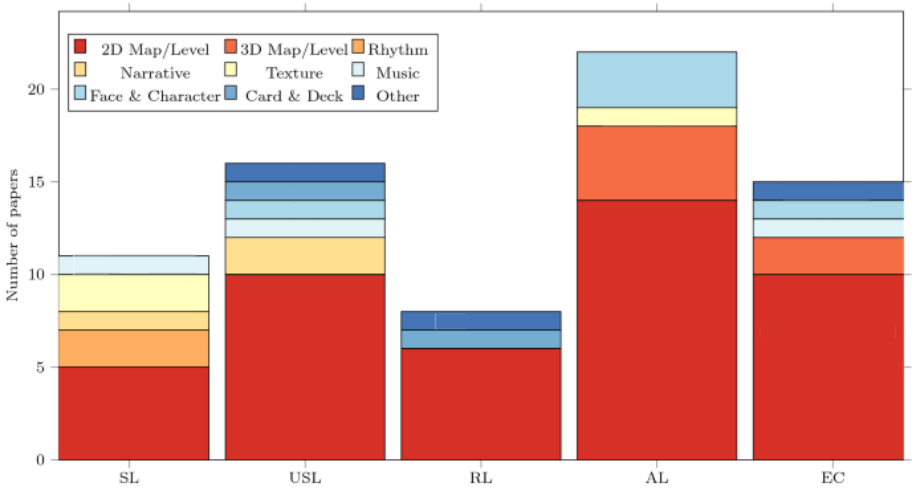


Figura 1.1: Gráfico que faz um levantamento de técnicas usadas para geração de conteúdo em jogos, onde AL equivale a *Adversarial Learning*, SL a *Supervised Learning*, USL a *Unsupervised Learning*, RL a *Reinforcement Learning* e EC a *Evolutionary Computing*. Fonte: adaptada de (Liu et al., 2020)

Grandes exemplos do uso de geração procedural são os jogos *Minecraft*, onde cada mapa é gerado em cima de um número aleatório, uma seed, tornando cada mundo diferente do outro - de forma tão diversificada que seria impensável um humano fazer isso -, *No Man's Sky* e o recente *Starfield*, onde planetas e biomas são gerados proceduralmente num ecossistema online multijogador, e jogos do gênero rogue, onde cada vez que o jogador perde o jogo, isto é, recebe uma quantia de dano à mais do que o suportável, pode melhorar sua personagem e mais uma vez enfrentar os desafios do início, porém, diferentes a cada jogada, como *Rogue Legacy* (1 e 2), *Enter The Gungeon* e *Hades*.

A área acadêmica também estuda a geração de conteúdo para jogos, muitos destes estudos ligados a geração de níveis através do aprendizado adversarial, conforme levantado em

(Liu et al., 2020) na **Figura 1.1**. Um modelo popular para geração de níveis para o jogo *Super Mario Bros.* foi proposto em (Volz et al., 2018). A grande inovação do artigo que o tornou popular no nicho não foi a arquitetura da GAN em si, mas sim o fato de utilizar o conceito de evolução de variável latente (LVE) para manipular as entradas do gerador de forma que o conteúdo produzido pelo modelo seguisse certo padrão através de um algoritmo genético, o CMA-ES, proposto inicialmente em (Hansen et al., 2003). O artigo em questão treina o modelo neural proposto apenas com uma única fase do jogo *Super Mario Bros* e avalia a qualidade da fase com a função objetivo do algoritmo genético e também através de métricas de um agente A* jogando o nível gerado, porém não se preocupa em verificar se estas fases seguem ou não de forma suficiente o conjunto de fases conhecidas. Outra limitação de (Volz et al., 2018) é o uso apenas do CMA-ES, não se preocupando em outras formas de explorar o espaço latente, ou tentar uma abordagem multi-objetivo, algo sugerido no final do artigo como possível estudo futuro.

1.1 OBJETIVO

Este trabalho utiliza a rede geradora proposta em (Volz et al., 2018) e, além do conjunto de treinamento formado por uma única fase do jogo - modelo treinado este disponibilizada pelos autores - neste estudo o modelo defendido pelo artigo também foi treinado com as fases de cada um dos 8 mundos do jogo *Super Mario Bros* (agrupamentos de até três níveis, que tem como objetivo um aumento de dificuldade gradual) no intuito de avaliar tanto a qualidade, jogabilidade e diversidade das fases geradas em um conjunto maior de níveis quanto se as redes aproximam ou não de forma que seja possível diferenciar o conjunto de fases apresentados no treinamento do modelo gerador.

Além disto, foi avaliada a efetividade do algoritmo proposto para a evolução de variáveis latentes, o CMA-ES, em comparação com um algoritmo de otimização de partículas (PSO). Também houve a comparação com uma abordagem multi-objetivo através do algoritmo NSGA-II proposto em (Deb, 2002) otimizando diferentes funções tanto com relação a topologia do nível quanto às funções propostas em (Volz et al., 2018).

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo revisita os tópicos fundamentais deste trabalho, incluindo os princípios de redes neurais adversariais, evolução de variável latente e algoritmos genéticos para exploração do espaço latente com o exemplo do CMA-ES; explora a literatura no tópico de geração de conteúdo procedural utilizando aprendizado de máquina; e apresenta o principal artigo que guiou este trabalho, que utiliza uma DCGAN para gerar níveis do jogo *Super Mario Bros* (1985).

2.1 REDES NEURAIS GENERATIVAS ADVERSARIAIS

O paradigma do Aprendizado Adversarial e as Redes Neurais Generativas Adversariais (GANs) foram introduzidos em (Goodfellow et al., 2014) em 2014 como solução para o problema de estimação de máxima verossimilhança, dilema no qual o custo computacional e a complexidade são altos, podendo chegar a ser computacionalmente inviável atingir uma solução ótima por meios clássicos de busca. Enquanto a gama mais popular de modelos profundos na época de publicação do artigo consistiam de apenas uma rede neural de alta profundidade usada em classificação e, em certos casos, regressão, o aprendizado adversarial proposto pelo artigo usufrui de dois MLPs.

Um destes modelos é apelidado pelos autores de gerador e abreviado como G, enquanto o outro modelo é chamado de discriminador e abreviado como D. O modelo gerador representa uma função diferenciável responsável por aproximar a distribuição de seus dados de treinamento. Para gerar novos dados, a rede G projeta entradas ruidosas no espaço dos dados conhecidos, na intenção de aproximar o conteúdo gerado da distribuição dos conjuntos conhecidos.

Por sua vez, o MLP D tem como objetivo traçar um plano que separe as distribuições do conteúdo original das distribuições das projeções de G. A saída da rede discriminadora, no artigo, é descrita como um único escalar que representa a probabilidade de uma entrada não ser gerada. Essencialmente, G e D tornam-se antagônicas a partir do momento que suas respectivas funções de perda colocam como objetivo penalizar a outra. Para que isso seja possível, G é treinada para minimizar $\log(1 - D(G(z)))$ enquanto D é treinada para maximizar a probabilidade de acertar se uma entrada é sintética ou real. Este treinamento antagônico leva essas duas redes a disputarem um jogo minimax de dois jogadores. Alternando a otimização de D e G, a rede discriminadora fica mais perto da solução ótima, enquanto a MLP geradora muda aos poucos para sua distribuição aproximar cada vez mais da original e, por consequência, tornar a solução que o discriminador busca mais complexa de se aproximar.

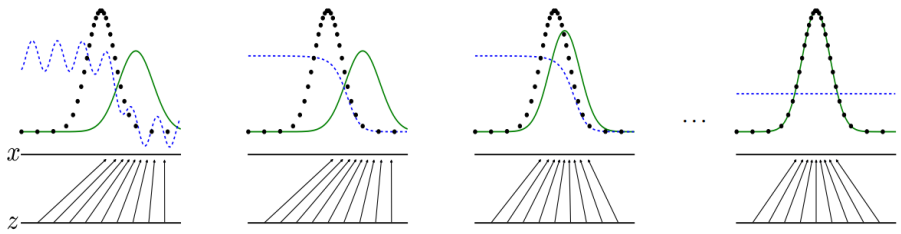


Figura 2.1: Representação do processo de aproximação de uma distribuição (linha pontilhada) pelo gerador (linha verde), enquanto o discriminador (linha azul) tenta separar as duas distribuições. Ao final do processo, o gerador aproxima completamente a distribuição dos dados, fazendo com que o discriminador não seja mais capaz de diferenciar as duas distribuições. Fonte: (Goodfellow et al., 2014)

2.2 CMA-ES

CMA-ES, a *Covariance Matrix Adaptation Evolution Strategy*, foi instanciada primeiro em (Hansen et al., 2003) e é um algoritmo derivado do paradigma da computação evolutiva, um ramo da inteligência computacional inspirado pela evolução biológica, ramo este que resolve problemas de otimização utilizando populações, meta-heurísticas e muitas vezes otimização estocástica. O CMA-ES é uma estratégia evolucionária que se comporta bem com vetores de números reais, que neste casos são as entradas do gerador, e sua principal característica é o uso de uma matriz de covariância estimada iterativamente por diferenças finitas.

Como está no seu nome, o algoritmo faz parte de um grupo de algoritmos evolucionários chamados *evolutionary strategies*, que é um grupo de algoritmos caixa preta caracterizados também pelo uso da ordenação por ranking. O CMA-ES tem se mostrado eficiente para otimizar problemas não-convexos e não-lineares em domínios contínuos.

O que este algoritmo faz é calcular uma Gaussiana com base na matriz de covariância estimada. Esta Gaussiana é traduzida numa hiper-elipse que se desloca de forma a contemplar as melhores soluções. Quanto mais distantes as melhores soluções estão do domínio dessa hiper-elipse, maior o *step size* dela, ou seja, seu raio. No começo, o *step size* costuma ser mais alto pois as melhores soluções conhecidas estão esparsas e a partir deste ponto de partida há a exploração global. Conforme a hiper-elipse fica melhor posicionada no domínio, as melhores soluções ficarão menos esparsas, fazendo com que o *step size* diminua proporcionalmente com a concentração da população. O **Algoritmo 1** ilustra um pseudo-código do algoritmo CMA-ES.

Algoritmo 1 CMA-ES

Inicialize a população: $\lambda > 0$ tal que $\lambda \in \mathbb{N}$
Inicialize o *step size*: $\sigma > 0$ tal que $\sigma \in \mathbb{R}$
Defina a média $m \in \mathbb{R}$ inicialmente aleatória
Defina o número de melhores candidados $\mu \in \mathbb{R}$ tal que $\mu > 0$
Defina os pesos: $W_1, W_2, \ldots, W_\mu \in \mathbb{R}$ tal que $\sum_i W_i = 1$
repeat
for all $x \in \lambda$ **do**
    Calcula a Gaussiana para x
end for
Ordena λ pelo fitness (função objetivo a ser minimizada) $\in \mathbb{R}$
Atribui a $X_1, X_2, \ldots, X_\mu$ os μ melhores candidados da iteração
$c \leftarrow$ Calcula_covariancia (λ)
$m \leftarrow \sum_{i=1}^{\mu} W_i \cdot X_i$
$\sigma \leftarrow$ Atualiza(σ)
until número de iterações $> max\_iter$

2.3 EVOLUÇÃO DE VARIÁVEL LATENTE (LVE)

LVE - *Latent Variable Evolution* - surgiu primeiro para buscar digitais específicas num certo espaço mapeado por uma GAN em (Bontrager et al., 2017). Como dito anteriormente, a rede neural geradora de uma GAN mapeia entradas ruidosas para um espaço. Estas entradas ruidosas também são chamadas de variáveis latentes, ou seja, variáveis que são difíceis ou impossível de controlar e interpretar antes de um resultado concreto.

A partir disso, a técnica de LVE tem como ideia manipular as entradas ruidosas de um modelo para que sua saída se encaixe dentro de certas especificações, ou seja, adiciona ao conteúdo gerado restrições que até o momento atual não são controláveis a nível de treinamento, pois uma rede geradora tem como única função reproduzir a distribuição de seus dados de teste. Em (Bontrager et al., 2017), a técnica de LVE é utilizada para a quebra de identificação de digitais. A técnica consiste de duas etapas: primeiro, treinar uma rede neural para gerar o conteúdo alvo e, após isso, usar um algoritmo para buscar no espaço latente uma entrada que gere uma saída dentro das condições especificadas.

O algoritmo utilizado por (Bontrager et al., 2017) para busca no espaço latente é o já explicado anteriormente CMA-ES. A justificativa do uso do CMA-ES pelo autores é que a função objetivo é não contínua e, pela natureza de redes convolucionais, também não é linearmente separável.

2.4 GERAÇÃO DE CONTEÚDO PROCEDURAL USANDO APRENDIZADO DE MÁQUINA (PCGML)

Summerville et al. (2018) define *Procedural Content Generation Via Machine Learning* como "geração de conteudo de jogos usando modelos de *machine learning* em conteúdos existentes". Segundo (Summerville et al., 2018), na área busca-se aumentar o valor de *replay* dos jogos, reduzir o custo e o esforço de produção, diversificar estética e otimizar o espaço de armazenamento. Em âmbitos acadêmicos também pode-se dizer que a área de PCGML explora novas maneiras de experenciar os jogos, além de ser um desafio que estimula a criatividade computacional e o entendimento de *game design* pela construção de modelos.

2.5 NSGA-II

Em 2002 foi proposto a evolução do *nondominated sorting genetic algorithm* (algoritmo genético de ordenção não-dominada), o NSGA-II (Deb, 2002), que hoje em dia conta com 37971 citações (Connected Papers, acesso em 05/11/2023). Este algoritmo foi proposto na intenção de contornar os três principais problemas dos algoritmos evolucionários apontados na época:

- Alta complexidade computacional (especificamente $O(MN^3)$, onde M é o numero de objetivos e N o tamanho da população);
- A falta de elitismo;
- A necessidade de especificar parâmetros, como o caso do NSGA proposto em (Srinivas e Deb, 1994);

E sua implementação nos permite:

- Preservar a diversidade;
- e enfatizar soluções não dominadas.

Para se entender as soluções não dominadas, é importante ententer o que é dominância entre soluções. A definição de dominância se dá que uma solução x_1 domina x_2 se x_1 não é pior que x_2 em todos os objetivos e é melhor que x_2 em pelo menos um objetivo. Quando não se há soluções que dominam x, x é uma solução não dominada. O conjunto de soluções não dominadas são chamados de fronteira não dominada. As projeções pertencentes a fronteira não

dominada se aproximam mais do ótimo em pelo menos um objetivo, mas podem ser piores do que outras da mesma fronteira em outros objetivos: essa troca entre os pontos não dominados fazem os algoritmos se interessarem em encontrar uma variedade da soluções antes de tomar uma decisão final.

Os pontos não dominados por nenhuma solução no espaço de soluções Z são os pontos ótimos de Pareto, enquanto o vetor de variáveis que projetam tais pontos são chamadas de soluções ótimas de Pareto. O objetivo do NSGA-II e outros algoritmos multi-objetivos, são, portanto, encontrar o máximo de pontos ótimos de Pareto, para que algum algoritmo de decisão escolha a solução com o melhor balanceamento entre os objetivos.

Outra importante propriedade utilizada pelo NSGA-II são as classes das fronteiras de Pareto. A fronteira de classe 1 são os já explicados pontos não dominados. A seguir, os pontos que não são dominados por nenhum ponto que não os ótimos de Pareto pertencem à fronteira de classe 2. Já os não dominados por nenhum a não ser os pontos das classes anteriores são a fronteira de classe 3, assim sucessivamente

Tendo populaçoes de tamanho N, o NSGA-II recombina tal população numa população de tamanho 2N e a ordena baseado no ranking da fronteira de Pareto, priorizando soluções da fronteira classe 1 e completando com as fronteiras subsequentes. Quando a fronteira F_g não cabe mais inteira na próxima população, são selecionadas os indivíduos com maior *crowding distance*, que é a área/volume (dependente do tamanho do espaço objetivo Z do problema) ao seu redor sem nenhuma solução presente (cabe enfatizar que as funções objetivo são normalizadas para que nenhuma seja mais priorizada pelo algoritmo do que outra). Este operador permite escolher as soluções que estão mais distantes das outras, por consequência também sendo um preservador de diversidade por si só.

2.5.1 Decisão entre soluções

Um dos dilemas de algoritmos multi-objetivo é projetado numa área chamada de MCDM (*multiple criteria decision making*), em que deve-se decidir entre as soluções qual é a melhor. Os experimentos deste trabalho seguem a ideia de (Rao, 2010) de *compromise programming* (CP) para este propósito conforme discutido em (Ringuest, 1992).

Assumindo a existência de uma solução ideal e a conversão próxima à fonteira de Pareto, o suposto ponto ideal é projetado levando em conta as soluções não dominadas existentes (ou seja, os menores valores dos objetivos se a função é minimizada, ou os maiores caso a função seja maximizada). Uma vez feita essa projeção, o ponto escolhido é o de menor distância deste ponto.

2.6 PSO

Otimização por enxames de partículas é um algoritmo bioinspirado proposto por (Kennedy e Eberhart, 1995), em que sua inspiração provém da navegação em bando de pássaros. Segundo os autores, é um método de otimização para funções contínuas e não-lineares que simula um modelo social simplificado. O algoritmo instância uma população candidata e a move pelo domínio de acordo com fórmulas matemáticas que não possuem alto custo computacional associado ao seu cálculo em relação a posição e velocidade da partícula (partícula, neste caso, equivale a uma solução candidata).

Além de cada uma das partículas se deslocar na sua direção atual no espaço de soluções (esta direção é atualizada a cada iteração do algoritmo), há também deslocamentos em direção ao melhor ponto observado pela particúla individualmente e para a melhor solução encontrada entre todas as partículas, que também pode sofrer alteração se alguma solução melhor for encontrada

durante a iteração. A velocidade de deslocamento da população candidata também varia, tanto coletivamente - reduzindo de acordo com o número de iterações, para favorecer primeiro a exploração global e, após isto, a exploração local - quanto individualmente, levando em conta a distância para a melhor solução individual e global.

A meta-heurística associada ao PSO faz pouca ou nenhuma suposição do problema que será otimizado. Outra vantagem é que o algoritmo não utiliza otimização por gradiente como muitos dos algoritmos de computação evolutiva, o que significa que a função objetivo não precisa ser diferenciável.

2.7 ALGORITMO A*

O algoritmo A* é uma extensão do Algoritmo de Djikstra que utiliza alguma meta-heurística para guiar a busca pelo caminho mais curto ao objetivo. Para que o algoritmo funcione corretamente, a heurística definida para auxiliar a busca deve ser admissível, isto é, o custo estimado para atingir o estado final em relação ao estado atual deve ser menor ou igual ao custo verdadeiro para atingir o estado final do estado atual.

A busca A* possui múltiplas aplicações, entre elas a busca por caminhos viáveis em videogames. Até os dias de hoje, o mais popular algoritmo para o teste de jogabilidade de níveis do jogo *Super Mario Bros* é o agente A* proposto em (Togelius et al., 2010). Neste agente específico, o custo é calculado pela distância resultante da ação em relação ao fim do nível do jogo, porém a heurística sempre leva em consideração o agente em velocidade máxima, o que pode não ser verdade.

2.8 MARIOGAN

Em 2018 foi publicado o artigo que deu origem ao que outros entusiastas do estudo da PCGML chamam de MarioGAN (Volz et al., 2018). O artigo aborda o uso de GANs, mas não foca na arquitetura da rede geradora ou discriminadora, mas sim utiliza a ideia de LVE para geração de níveis procedurais no jogo *Super Mario Bros*. Neste artigo, seguindo a ideia de (Bontrager et al., 2017), o CMA-ES é utilizado para adicionar alguma restrição ao espaço latente do gerador, sendo testadas quatro funções objetivo diferentes, duas delas relacionadas à disposição dos blocos da fase e outras duas em relação ao teste das fases pelo agente A* vencedor da competição Mario AI de 2009 (Togelius et al., 2010) de autoria de Robin Baumgarten, porém com poucos exemplos deste segundo caso.

Para representar as fases, os autores utilizaram a representação do jogo *Super Mario Bros* disponível no repositório do *github* aberto da Video Game Level Corpus(VGLC) (Summerville et al., 2016), que, entre suas particularidades, simplifica a representação do jogo ao ignorar níveis específicos, onde a física do jogo diverge do padrão seguido pela maioria das fases, e também representa todos os inimigos da mesa maneira, tornando a tarefa possivelmente mais simples. Nesta representação, cada tipo de bloco é mapeado para um caracter específico, transformando a imagem de um nível em uma sequência de caracteres que indicam um bloco específico. Uma vez que o artigo implementa a arquitetura a partir da biblioteca *Pytorch*, esta representação textual precisou ser remapeada para uma representação numérica, conforme mostrado pela **Figura 2.2**.

A GAN convolucional profunda (DCGAN) utilizada no artigo é baseada na Wassertein Gan apresentada em (Arjovsky et al., 2017), em que o discriminador utiliza convoluções com deslocamento (*strided convolutions*), enquanto o gerador utiliza convoluções com deslocamento fracionário (*fractional-strided convolutions*). A arquitetura em (Volz et al., 2018) difere da arquitetura de (Arjovsky et al., 2017) nos seguintes pontos:

Tile type	Symbol	Identity	Visualization
Solid/Ground	X	0	
Breakable	S	1	
Empty (passable)	-	2	
Full question block	?	3	
Empty question block	Q	4	
Enemy	E	5	
Top-left pipe	<	6	
Top-right pipe	>	7	
Left pipe	[	8	
Right pipe	]	9	

Figura 2.2: Representação dos elementos do jogo *Super Mario Bros* pela VGLC e pelos autores de (Volz et al., 2018), respectivamente. Fonte: (Volz et al., 2018)

- Tanto no discriminador quanto no gerador, é empregada uma camada de normalização de bloco (*batch normalization*) após cada camada convolucional;
- A função de ativação de todas as camadas (inclusive na de saída) é a ReLU (O discriminador, como em (Arjovsky et al., 2017), utiliza a Leaky ReLU como função de ativação em suas camadas).

Tanto o discriminador quanto o gerador são treinados com o otimizador RMSprop, com um *batch size* de tamanho 32 e a taxa de aprendizado de 0,00005 por 5000 iterações. A arquitetura é devidamente ilustrada na **Figura 2.3**.

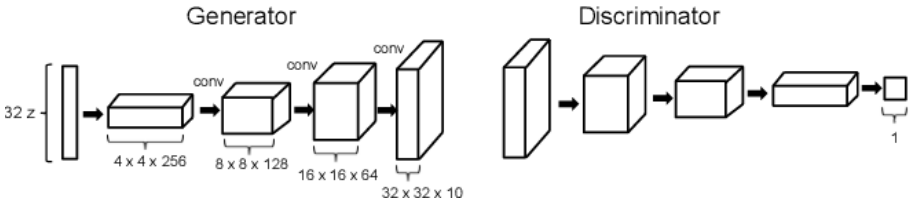


Figura 2.3: Representação da arquitetura da GAN utilizada no paper. Fonte: (Volz et al., 2018)

O gerador utilizado no artigo foi treinado apenas com o primeiro nível do jogo, na intenção de mostrar como a LVE e a GAN treinada poderiam, em conjunto, gerar níveis diferentes e jogáveis. Outra escolha interessante do artigo da MarioGAN (Volz et al., 2018) é que, para gerar múltiplas amostras com apenas uma fase, a representação numérica foi separada com base em uma janela deslizante com dimensões de 14 de altura e 28 de largura, com passo de tamanho um.

Uma noção geral do processo executado por (Volz et al., 2018) está devidamente traduzido na **Figura 2.4**. Este método é dividido em duas etapas básicas distintas: primeiro, a WGAN é treinada de maneira não-supervisionada para gerar níveis do *Super Mario Bros*; após isso, o CMA-ES é utilizado para buscar no espaço latente da GAN soluções que minimizem a função objetivo escolhida, ou seja, alterar a entrada do modelo para satisfazer propriedades específicas. Embora o artigo se baseie apenas em níveis do jogo *Super Mario Bros*, os autores clamam que o modelo é suficientemente bom para generalizar qualquer jogo com representação compatível.

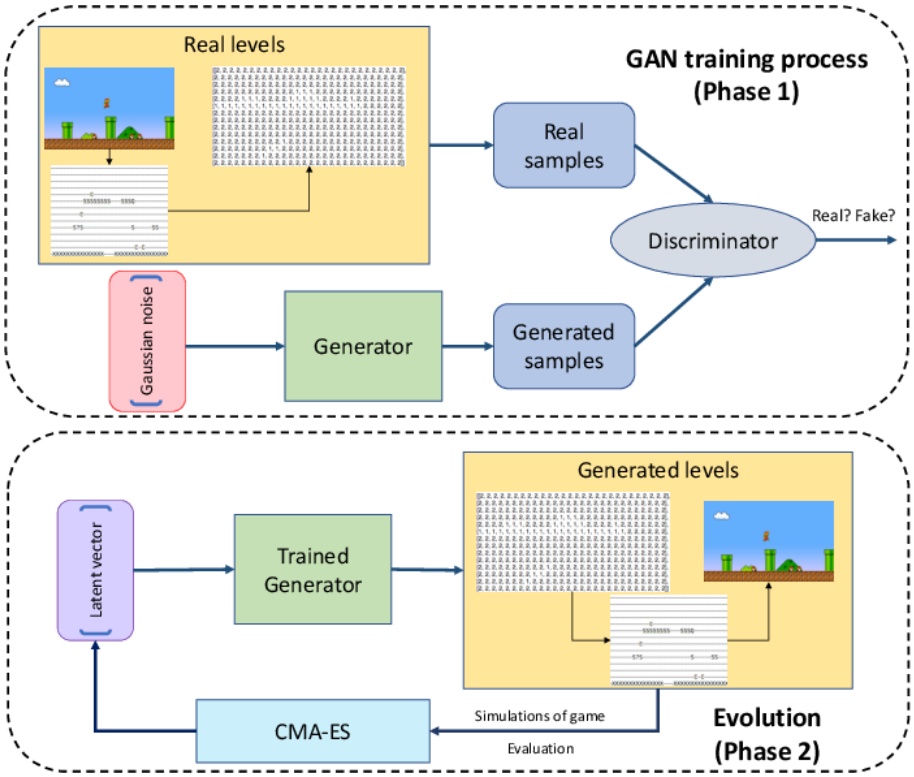


Figura 2.4: Representação do processo de treinamento e de LVE. Fonte: (Volz et al., 2018)

As equações (2.1) e (2.2) mostram as funções objetivo minimizadas pelo CMA-ES em relação a topologia do nível usadas no artigo, em que g e t são respectivamente a fração da quantidade de blocos de chão na ultima linha da matriz que representa a fase e a fração desejada, e n o número de inimigos.

$$F_1 = \sqrt{(g-t)^2} \tag{2.1}$$

$$F_2 = F_1 + 0.5 \cdot (20.0 - n) \tag{2.2}$$

No artigo, a ideia apresentada é que a dificuldade aumente gradativamente conforme a progressão no eixo x, lembrando que o objetivo do jogo é chegar ao fim da fase, ou seja, no último ponto do eixo x. Para isso, os autores concatenam no eixo x dois níveis otimizados com F_1 e $t = 1$, um nível otimizado com F_1 e $g = 0.7$, e dois níveis otimizando F_2 com $g = 0.7$. Um exemplo desta junção está na **Figura 2.5**.

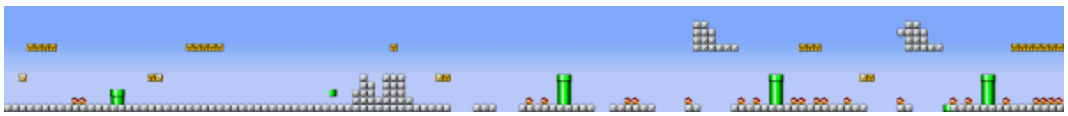


Figura 2.5: Nível gerado pelos autores, com progressão de dificuldade no eixo x. Fonte: (Volz et al., 2018)

Em outros experimentos, os autores usam outras duas funções objetivo, funções estas relacionadas ao teste pelo agente A* vencedor da competição da mario AI de 2009, conforme as

equações (2.3) e (2.4), onde p é a fração da fase em que o agente passou, e #jumps a quantidade de ações de pulos.

$$F_3 = \begin{cases} -p, & \text{se } p < 1, \\ -p - \#jumps, & \text{se } p = 1. \end{cases} \tag{2.3}$$

$$F_4 = \begin{cases} -p + 60, & \text{se } p < 1, \\ -p + \#jumps, & \text{se } p = 1. \end{cases} \tag{2.4}$$

Na verdade, F_3 maximiza os pulos e recompensa fases que o agente é capaz de completar, enquanto F_4 também recompensa fases jogáveis, mas minimiza pulos. A **Figura 2.6** ilustra alguns desses níveis gerados.

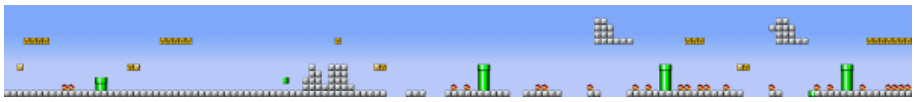


Figura 2.6: Nível gerado pelos autores, de acordo com a performance do agente. A letra a utiliza F_3, enquanto a parte b utiliza F_4. Fonte: (Volz et al., 2018)

Lembrando que não foram feitas abordagem multi-objetivo utilizando F_1 ou F_2 e F_3 ou F_4 no artigo.

2.9 CONSIDERAÇÕES FINAIS

Neste capítulo, foram apresentados os conceitos de GAN, PCGML e LVE, além de três algoritmos bioinspirados - CMA-ES, PSO e NSGA-II - que se relacionarão diretamente aos demais conceitos no decorrer do trabalho. Também foi apresentado o algoritmo A* que será utilizado para teste e otimização das fases, além de uma visão ampla de um dos principais artigos da área de PCGML, artigo em que a rede proposta ficou conhecida como MarioGAN. No próximo capítulo, serão visitados alguns outros artigos da área, muitos dos quais utilizam de alguma forma os conceitos aqui apresentados.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta uma visão geral de estudos existentes em relação ao uso de redes generativas adversariais para geração de fases e também do uso da exploração de variáveis latentes para guiar o conteúdo gerado. Ele revisa a definição de geração de conteúdo procedural, apresenta sistemas utilizados para geração de conteúdo e discute modelos de geração de conteúdo relevantes para a literatura.

3.1 GERAÇÃO DE CONTEÚDO PROCEDURAL VIA MACHINE LEARNING UTILIZANDO REDES GENERATIVAS ADVERSARIAIS

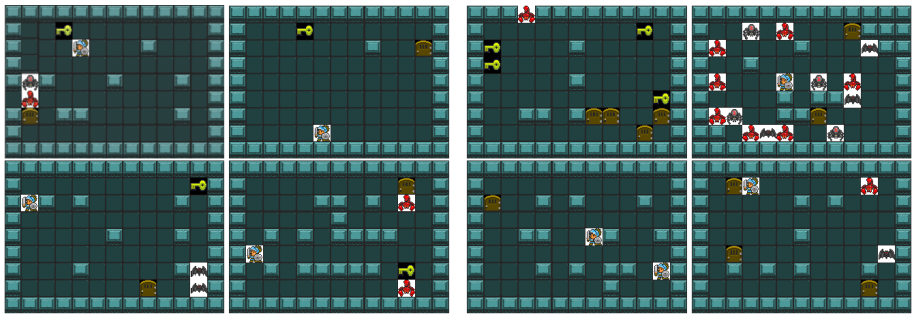
Antes mesmo do machine learning chegar à área de geração procedural de conteúdo, vários métodos baseados em busca como n-gramas e cadeias de Markov já eram empregados como levantado em (Togelius et al., 2011). Neste artigo de certa popularidade, PCG é definida como "criar conteúdo para jogos automaticamente, através de meios algorítmicos". Já em (Hendrikx et al., 2013), os autores definem PCG-G (geração de conteúdo procedural para jogos) como "a aplicação de computação para gerar conteúdo para jogos, distinguindo instâncias interessantes entre as amostras geradas, e selecionar instâncias interessantes em prol dos jogadores". Embora o termo não tenha se popularizado, (Hendrikx et al., 2013) comenta muito bem vários dos desafios dessa área, com ênfase em como julgar o valor técnico e cultural do conteúdo gerado.

Alguns exemplos de PCGML são o *Bardo Composer*(Ferreira et al., 2020) que gera músicas para RPGs de Mesa, geração de personagens em jogos no estilo pixel-art em (Rebouças Serpa e Formico Rodrigues, 2019), ou até mesmo aplicações mais práticas, como a da desenvolvedora *CD Projekt RED*, que fez uma parceria com a empresa canadense *Jali Research* para, através de técnicas híbridas de inteligência artificial clássica e aprendizado de máquina, fazer com que a boca dos personagens se adapte à dublagem correspondente do diálogo, e o recente anúncio da desenvolvedora francesa *Ubisoft* do desenvolvimento de uma ferramenta para ajuda de construção de narrativas de jogos, a *Ubisoft Ghostwritter*.

Porém a geração de conteúdo procedural foi utilizada pela primeira vez pelo jogo de 1984 *Elite*, onde os desenvolvedores usavam uma sequencia definida de sementes aleatórias para gerar composições de planetas na intenção de economizar memória armazenando apenas um gerador ao invés dos planetas inteiros. De acordo com o gráfico da **Figura 1.1**, quando se fala de PCGML, há uma quantidade desproporcional de pesquisa em geração de níveis 2D em comparação aos demais tópicos. Além disso, (Liu et al., 2020) ilustra no mesmo gráfico o quanto o paradigma do aprendizado adversarial domina este tópico. Segundo os autores, os modelos adversariais mais populares neste campo seriam as GANs e suas variações.

Em (Giacomello et al., 2018), algumas GANs combinaram imagens e atributos topológicos e posicionais extraídos de conteúdo feito por humanos para gerar níveis do jogo *Doom*. Neste trabalho em específico, os níveis não foram testados para verificar se são jogáveis ou não. Já em (Torrado et al., 2019), além de propor uma nova arquitetura que acrescenta camadas de *embedding* e módulos de atenção chamada CESAGAN, a pequena quantidade de dados de treinamento do jogo *The Legend Of Zelda (1986)* foi expandida por uma técnica de *bootstrapping*. Alguns dos resultados de (Torrado et al., 2019) estão na **Figura 3.1**.

(Capps e Schrum, 2021) utiliza múltiplas GANs, um modelo o qual os autores apelidaram como MultiGAN, para gerar níveis melhores conectados do jogo *Megaman* (1987). Para evoluir os níveis a atingirem certa propriedade, ao invés do CMA-ES, os autores utilizam o NSGA-



(a) Playable Levels (b) Unplayable Levels

Figura 3.1: Fases geradas pela CESAGAN. Em a, níveis jogáveis e em b níveis injogáveis. Fonte: (Torrado et al., 2019)

II (*Non-Dominated Sorting Genetic Algorithm II*), um algoritmo multi-objetivo baseado em conceitos de Pareto. Além disso, (Capps e Schrum, 2021) utiliza um algoritmo A* para predizer os caminhos e ajudar na mutação das variáveis latentes.

GANs também são utilizadas para produzir proceduralmente outros conteúdos além de níveis 2D. (Hong et al., 2019) propões o uso de dois codificadores (*encoders*) distintos para estrutura, formato e cor, juntamente com discriminadores distintos para formato e cor para geração de silhuetas de personagens. (Jin et al., 2017) também utiliza redes generativas adversariais para gerar personagens, desta vez para o jogo *The Sims* (2000). Em outro exemplo, (Fadaeddini et al., 2018) utiliza GANs para geração de texturas em jogos.

Como já explicado anteriormente, (Volz et al., 2018) define uma DCGAN baseada na WGAN apresentada por (Arjovsky et al., 2017) treinada em um único nível do jogo *Super Mario Bros* e inova na forma de adicionar restrições a rede geradora após seu treinamento utilizando a técnica de LVE para adicionar restrições a saída do modelo. Neste artigo, o CMA-ES é o algoritmo utilizado para a busca no espaço latente e é aplicado após o treinamento da rede neural adversarial.

3.2 CONSIDERAÇÕES FINAIS

Neste capítulo, foram revisitados alguns trabalhos que utilizam conceitos de PCGML, muitos trabalhos estes que utilizam conceitos de (Volz et al., 2018), além de algumas aplicações práticas no início da sessão. Não se limitando a geração de níveis, o objetivo desta sessão foi justificar a importância do aprendizado adversarial na área de PCGML como um todo, área esta que vem recebendo certo volume de pesquisa, porém ainda recente. No próximo capítulo, será apresentada a proposta deste trabalho.

4 PROPOSTA

Este capítulo apresenta os métodos propostos para comparação entre os métodos de LVE, bem como os para avaliação do conteúdo gerado por redes treinadas em diferentes conjuntos. Ele apresenta a arquitetura do modelo neural adversarial utilizado, o conjunto de dados em que o modelo foi treinado e as funções objetivo utilizadas.

4.1 CONJUNTO DE TREINAMENTO

O jogo escolhido para o trabalho é o mesmo presente em (Volz et al., 2018), *Super Mario Bros*. Enquanto (Volz et al., 2018) propõe avaliar a diversidade de conteúdo gerado a partir do treinamento do modelo com um único nível (o primeiro), neste trabalho é avaliado o quão diversificados são os conteúdos gerados em conjuntos de granularidade diversa.

Como em (Volz et al., 2018), a representação utilizada é a mesma do repositório aberto da *Video Game Level Corpus* (Summerville et al., 2016), em que os blocos estão representados na **Figura 2.2**. Reproduzindo a divisão do artigo de inspiração, a representação textual é dividida por uma janela deslizante de 14 blocos/caracteres de altura e 28 blocos/caracteres de largura com passo de tamanho um. Além do modelo disponibilizado por (Volz et al., 2018), treinado em uma única fase, foram observados conjuntos de treinamento formados pelos oito mundos do jogo (agrupamentos de níveis com dificuldade gradual, proporcional ao número do mundo). A **Tabela 4.1** mostra, em sua totalidade, a quantidade de níveis processados em cada conjunto de treinamento, bem como a quantidade de janelas obtidas.

Mundo	Número de Níveis	Número de Janelas Deslizantes
Primeira Fase	1 Nível	173
Mundo 1	3 Níveis	473
Mundo 2	1 Nível	170
Mundo 3	2 Níveis	292
Mundo 4	2 Níveis	355
Mundo 5	2 Níveis	294
Mundo 6	3 Níveis	452
Mundo 7	1 Nível	149
Mundo 8	1 Nível	346

Tabela 4.1: Número de níveis e janelas deslizantes por Mundo

4.2 ARQUITETURA DO MODELO NEURAL

Neste trabalho, a arquitetura do modelo de (Volz et al., 2018) foi preservada, arquitetura esta devidamente ilustrada na **Figura 2.3**. A principal motivação para tal preservação é para que a comparação entre conjuntos de treinamento e métodos de busca em espaço latente seja justa. Embora seja possível trocar o modelo no fluxo de dados (assim como é possível trocar o algoritmo usado para LVE), este não é o foco deste trabalho. O código utilizado no modelo é o mesmo disponibilizado por (Volz et al., 2018), com sua implementação com auxílio da biblioteca *PyTorch* (Paszke et al., 2019).

4.3 ALGORITMOS PARA LVE

Este estudo realiza a comparação de três diferentes algoritmos para LVE. Primeiro, o modelo neural generativo adversarial é devidamente treinado em algum conjunto de níveis humanos do jogo *Super Mario Bros*. Após isso, um dos métodos de evolução de variável latente é aplicado na intenção de guiar a geração de conteúdo para atender propriedades específicas.

Os algoritmos utilizados para a evolução de variável são: o CMA-ES, como em (Volz et al., 2018), o PSO como inovação, e o NSGA-II, que foi usado em (Torrado et al., 2019). A motivação para a escolha desses três algoritmos foi, respectivamente, manter o algoritmo utilizado em (Volz et al., 2018) juntamente com seu paradigma (um algoritmo genético), testar a otimização por enxame de partículas por ser rápida e de menor complexidade computacional, e também avaliar uma abordagem multi-objetivo como sugerido em (Volz et al., 2018). A **Tabela 4.2** sintetiza os hiperparâmetros destes três algoritmos, onde *popsize* é o tamanho da população, *sigma0* é o *step size* inicial do CMA-ES, *n_gen* o número de gerações que o algoritmo irá trabalhar, *eliminate_duplicates* se o algoritmo irá considerar soluções duplicadas, e *iterations* a quantidade de gerações do CMA-ES.

Algoritmos	Hiperparâmetros
CMA-ES	*Popsize = 28, sigma0 = 0.5, iterations = 50* (Para a comparação de tempo)
PSO	*Popsize = 28, eliminate_duplicates=True, n_gen = 50*
NSGA-II	*Popsize = 28, eliminate_duplicates=True, n_gen = 50*

Tabela 4.2: Configurações dos Algoritmos com seus Hiperparâmetros

Para o CMA-ES, a biblioteca utilizada foi a *CMA* (Hansen et al., 2019). Já para o PSO e o NSGA-II, foram usadas as implementaçõs do *framework* pymoo (Blank e Deb, 2020). Em termos práticos, o que acontece neste e em outros experimentos que não utilizam o CMA-ES é que, levando em consideração a **figura 2.4** como exemplo, o CMA-ES é trocado pelo PSO ou NSGA-II e são feitas as devidas comparações.

4.4 FUNÇÕES OBJETIVO

Neste trabalho, muitas funções objetivo foram otimizadas. Foram minimizadas as funções das equações (2.1) e (2.2) seguindo a proposta de progressão de dificuldade em (Volz et al., 2018), e o desempenho do CMA-ES e do PSO foram comparados. Também foram feitas algumas rodadas de otimização destas funções em conjunto com a presente na equação (2.3) utilizando NSGA-II, para avaliar o desempenho geral, conforme sugerido por (Volz et al., 2018).

Além das funções das equações (2.1), (2.2) e (2.3), também foi utilizada outra função objetivo para comparar a topologia das fases. Esta função está implementada no repositório aberto do *Github* dos autores de (Volz et al., 2018), mas não é mencionada no artigo e é relacionada ao número de blocos de um certo tipo do conteúdo gerado, seguindo a equação (4.1).

$$\frac{(100 - (\sum_{i=1}^{14} \sum_{j=1}^{28} f(x_{ij}, b)))}{100} \quad \text{tal que} \quad f(x,b) = \begin{cases} 1 & \text{se } x = b \\ 0 & \text{caso contrário} \end{cases} \tag{4.1}$$

Onde i é o valor de linha e j o da coluna da matriz que representa um nível, x_{ij} o valor numérico presente nesta posição e b o valor numérico a ser maximizado, ou seja, a representação

numérica do bloco a ser maximizado. Os valores de b utilizados foram 6 (um pedaço do cano) e 5 (número de inimigos).

Esta função é interessante pois sua interpretação tanto em valores numéricos quanto visualmente não é complexa: ela será menor quanto maior a quantidade de um certo elemento. Fases com mais inimigos, por exemplo, terão menores valores da função, algo que pode ser dito de visualizar um nível gerado. Além disso, maximizar um elemento no nível pode gerar fases diferentes de acordo com o elemento: fases com mais canos, por exemplo, serão naturalmente diferentes que fases com mais inimigos.

4.5 AVALIAÇÃO DE CONTEÚDO

Como devidamente apontado por (Hendrikx et al., 2013), uma das maiores dificuldades da PCGML é avaliar o conteúdo gerado proceduralmente. Neste trabalho, foram propostos três métodos de avaliação:

- Observação dos valores das funções objetivo utilizadas na técnica de LVE;
- Um classificador para avaliar se os geradores seguem suficientemente um padrão;
- Um agente para testar a jogabilidade dos níveis.

Para que cada teste deste seja em quantidades suficientes, foram avaliadas múltiplas execuções das duas fases propostas em (Volz et al., 2018) com várias funções objetivo e conjunto de treinamento, com exceção do NSGA-II otimizando a função da equação (2.3), pois o uso do agente escolhido dentro de um algoritmo genético se mostrou muito custoso no quesito tempo de execução: devido a isto, foram realizadas poucas execuções neste caso específico.

4.5.1 Classificação de níveis

A primeira abordagem para a classificação de níveis foi o uso de um SVC (sob a idéia de ser um conceituado método de classificação dito de bom desempenho para problemas desbalanceados de várias dimensões) de classe única treinado em todas as janelas deslizantes geradas a partir dos níveis existentes. Uma vez que esse método mostrou-se insuficiente por classificar um pouco menos da metade do conjunto de treinamento como anomalia, foi treinado um outro SVC buscando separar os 8 mundos mais uma classe extra de matrizes 14 X 28 com números aleatórios entre 0 e 9, totalizando 9 classes. A performance deste classificador não ultrapassou 55% de taxa de acerto.

Por fim, uma rede neural simples, codificada em *PyTorch* foi treinada em cima das mesmas janelas deslizantes e sua performance foi julgada suficiente para avaliar o padrão de uma fase. Este modelo possui apenas duas camada convolucionais com 32 neurônios, uma camada *maxpooling* e uma camada *fully connected*, com 64 neurônios, finalizando com uma camada de ativação softmax de 9 classes.

Para este treinamento, 80% do conjunto de dados foi usado para treino, 10% para validação e 10% para teste. Os resultados do teste estão presentes na **Tabela 4.3**, resultado este obtido em um único treinamento com todas as classes.

Para os testes de classificação, este modelo foi empregado em torno das saídas do modelo levando em conta as entradas geradas.

Mundo	Precisão	Recall	F1-Score
Lixo	1.00	1.00	1.00
Mundo 1	0.92	0.90	0.91
Mundo 2	1.00	1.00	1.00
Mundo 3	1.00	1.00	1.00
Mundo 4	1.00	1.00	1.00
Mundo 5	0.73	0.73	0.73
Mundo 6	0.96	1.00	0.98
Mundo 7	1.00	1.00	1.00
Mundo 8	1.00	0.97	0.98
Total Médio	0.95	0.95	0.95

Tabela 4.3: Resultados de Precisão, Recall e F1-Score por Mundo no teste

4.5.2 Jogabilidade e performance de jogo

Em 2009, em associação a *IEEE Games Innovation Conference* e a *IEEE Symposium on Computational Intelligence and Games*, ocorreu a *Mario AI Competition*, documentada em (Togelius et al., 2010). Até hoje, um dos agentes mais utilizados para testar a jogabilidade de níveis gerados do jogo *Super Mario Bros* é o agente A* de autoria de Baumgarten.

Este agente define o jogo como estado atual e leva em conta a velocidade do jogador para os estados próximos. Além do estado próximo, a heurística leva em conta que o agente está sempre na velocidade máxima, o que a torna nem tanto admissível, sendo uma relaxação segundo o autor já que esta heurística em alguns casos apontados em (Togelius et al., 2010) pode levar o agente A* a considerar que um estado leve a perda do jogo sem ser necessariamente verdade. Tanto em (Volz et al., 2018) quanto em (Togelius et al., 2010) é dito que o agente performa em níveis sobre-humanos, agente este que é utilizado até os dias de hoje, como em (Sudhakaran et al., 2023).

Seguindo os trabalhos já existentes, a jogabilidade das fases foi testada com o agente A* de Baumgarten, assim como este agente foi utilizado para o uso da equação (2.3) nos experimentos multi-objetivo. Alguns experimentos mostraram, porém, que este algoritmo é não determinístico, algo não mencionado por (Togelius et al., 2010) nem por (Volz et al., 2018).

A implementação do *Super Mario Bros* utilizada é a mesma de (Togelius et al., 2010), adquirida através do repositório dos autores no *Github*. Esta implementação é feita em *Java*. O código original foi levemente modificado, alterando um valor booleano para redirecionar a jogabilidade ao agente (que está também implementado neste repositório) e retirar a demonstração visual e o limite de quadros por segundo. Esta implementação em Java adquire o gerador de um código *Python*. Este código *Python* também foi modificado de acordo com o gerador utilizado (que varia entre experimentos).

4.6 CONSIDERAÇÕES FINAIS

Este capítulo sintetiza a ideia e o objetivo de cada experimento proposto, apresenta o conjunto de treinamento, as funções utilizadas dentro do LVE, as comparações entre os métodos de manipulação do espaço latent, e como serão avaliados estes resultados. No próximo capítulo, os resultados serão apresentados.

5 RESULTADOS

Neste capítulo, são apresentados os resultados dos testes propostos no capítulo anterior. Com análises tanto visual quanto em tabelas e gráficos, foram aplicadas técnicas de LVE utilizando diversas funções objetivo no espaço latente da GAN, modelo este que foi treinado com múltiplos conjuntos. Também vale lembrar que três tipos de algoritmos foram utilizados (CMA-ES, PSO e NSGA-II) e, além das funções objetivo, os níveis gerados também foram avaliados por um classificador neural simples e o algoritmo A* de Baumgarten.

5.1 COMPARAÇÃO DE TEMPO E DESEMPENHO ENTRE CMA-ES E PSO

Foi realizada uma comparação entre o PSO e o CMA-ES nos quesitos tempo de execução e desempenho, com ambos os algoritmos limitados a 50 iterações. Neste experimento, o conjunto de treinamento do gerador foi composto apenas por janelas geradas pelo processamento da primeira fase, para devidamente comparar o CMA-ES manipulando um gerador treinado com o conjunto de treinamento presente no artigo com o PSO executando a mesma função.

A função objetivo otimizada foi a da equação (2.2), com $t = 0.7$. Foram realizadas 200 iterações por método de LVE. Os resultados estão presentes na **Tabela 5.1**, com média e desvio padrão.

Método	**Média da Fitness**	**Desvio Padrão (Fitness)**	**Média do Tempo**	**Desvio Padrão (Tempo)**
CMA-ES	**10.0**	0	8.50	2.69
PSO	10.2	0.143	**2.08**	0.615

Tabela 5.1: Comparação de Tempo e Fitness do CMA-ES e PSO

Os resultados são interessantes. Primeiro, vale notar que o CMA-ES obteve um desvio padrão de 0, com todas suas execuções gerando o mesmo valor da função objetivo minimizada. Em caso de dúvida, algumas destas fases foram checadas e, de fato, as entradas - assim como as fases geradas - são diferentes, sendo pontos do espaço latente razoavelmente distantes entre si. Mais interessante que isso é notar como é pequena a diferença, algo que apenas um teste estatístico poderia dizer com toda certeza se é significativa ou não. Fato é que cada execução do CMA-ES levou, em média, 8.5 segundos, enquanto as execuções do PSO levaram apenas 2.08 segundos. Isto leva a crer que o uso de um algoritmo que demanda menos recursos computacionais, como o de otimização de partículas, é relevantemente mais rápido que o CMA-ES e alcança marcas parecidas ou até mesmo superiores.

5.2 NÍVEIS EM PROGRESSÃO DE DIFICULDADE

Neste experimento, nós comparamos o desempenho e a variedade de fases geradas dos métodos de LVE CMA-ES e PSO utilizando as funções das equações (2.1) e (2.2) da mesma forma sugerida no artigo: concatenando duas janelas geradas minimizando a função da equação (2.1) com $t = 1$, uma com $t = 0.7$, e outras duas utilizando a equação (2.2) com $t = 0.7$.

Neste experimento, foram considerados três diferentes conjuntos de treinamento: o conjunto gerado pelo processamento apenas da primeira fase - o mesmo de Volz et al. (2018) - e

os gerado pelo processamento das três fases do mundo 1 e 6. Para cada uma das cinco partes foram geradas 100 instâncias, totalizando 500 fases por método de LVE , com 300 para cada pedaço das fases. A **figura 5.1** mostra os resultados desta comparação entre o CMA-ES e o PSO para esta aplicação específica.

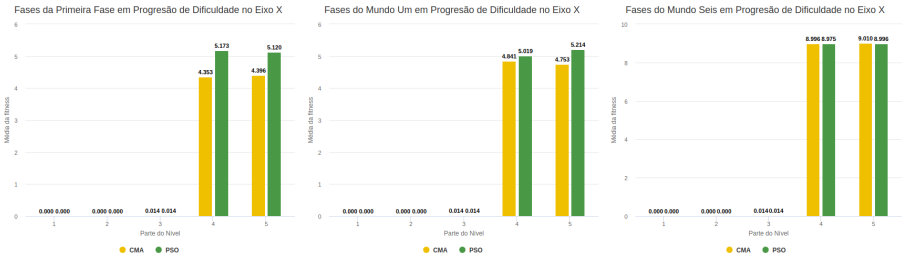


Figura 5.1: Gráficos com as médias das fitness, para os três conjuntos de treinamento especificados

Nota-se, neste gráfico, que as primeiras duas partes (que minimizam F_1 com $t = 1$) conseguem chegar a 0, que é o menor valor possível de F_1, sem dificuldades. Isto leva a crer que gerar níveis em que a matriz correspondente possui o bloco "0"em todas as posições da sua última linha - isto é, todo o chão da fase possui blocos - não é uma tarefa complexa para as técnicas de LVE, seja o algoritmo usado o PSO ou o CMA-ES. Uma vez que há muitas conversões neste primeiro caso, torna-se ainda mais interessante a ideia de minimizar simultaneamente F_3 numa abordagem multi-objetivo para tornar os níveis mais interessantes, forçando o agente testador a realizar mais ações de pulo, ação esta que pode ser relacionada com dificuldade.

Quanto a parte três, que minimiza F_1 com $t = 0.7$, os três conjuntos de treinamento apresentaram a mesma média para ambos os algoritmos utilizados para a técnica de LVE, o que indica um desempenho semelhante de ambos. Quando se fala da quarta e quinta parte da fase, partes estas que a função objetivo otimizada é F_2 com $t = 0.7$, o CMA-ES possui um desempenho 16% melhor no conjunto de treinamento formado pela primeira fase, 7% melhor no conjunto formado pelo processamento do primeiro mundo, e 0.02% pior no terceiro conjunto de treinamenjto em comparação com o PSO. Estes resultados não são expressivos o suficiente para afirmar qual técnica é melhor, levando em conta que o PSO executa em média, segundo o experimento da seção 5.1, pouco mais de 4 vezes mais rápido que o CMA-ES. Os desvios padrões do PSO também são maiores que o CMA-ES neste caso (no mundo 1, por exemplo, o desvio padrão da média da função objetivo das execuções do PSO para F_2 é de 0.425, enquanto o desvio padrão do CMA-ES é de 0.585), o que pode ser um indicativo de algumas execuções do PSO chegarem em soluções até melhores que o CMA-ES.

As figuras **figuras 5.2, 5.3** e **5.4** exemplificam níveis gerados com esta técnica utilizando os conjuntos de treinamento da primeira fase, do mundo um e do mundo seis, respectivamente. Interessante notar que o nível escolhido da primeira fase não parece ser jogável, já que possui um espaço muito grande sem blocos. Outra coisa interessante é que domina-se a quantia de inimigos sobre os 70% de blocos de chão nas fases das figuras **5.2 e 5.3**, enquanto isso não acontece na figura **5.4**, onde há poucos inimigos, mas há falhas nos blocos da linha mais baixa da fase como o esperado.

5.2.1 Abordagem multi-objetivo

No artigo de (Volz et al., 2018), há a sugestão de utilizar algum método multi-objetivo para a evolução de variáveis latentes, juntando as funções das equações (2.1) e (2.2) para serem utilizadas em conjunto a função da equação (2.3). O motivo para tal sugestão é que as primeiras duas funções levam em conta apenas a estrutura geral do nível com relação a chão e inimigos,

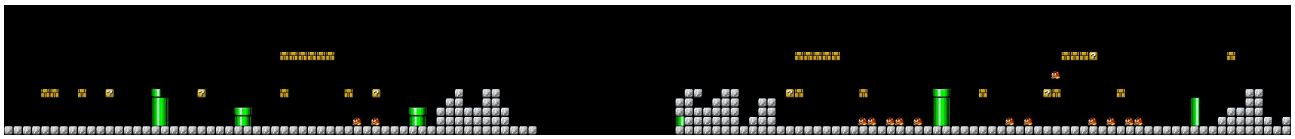


Figura 5.2: Nível gerado pela GAN treinada com a primeira fase, utilizando a técnica de LVE e funções objetivo sugeridas em Volz et al. (2018)

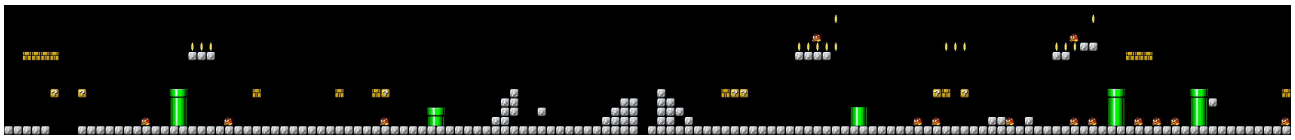


Figura 5.3: Nível gerado pela GAN treinada com as fases do mundo um, utilizando a técnica de LVE e funções objetivo sugeridas em Volz et al. (2018)

Figura 5.4: Nível gerado pela GAN treinada com as fases do mundo 6, utilizando a técnica de LVE e funções objetivo sugeridas em Volz et al. (2018)

com manipulação do parâmetro t para induzir uma dificuldade gradativa. O problema é que os níveis com melhores valores das funções de fitness, especialmente com a equação (2.2) como função minimizada (que diminui a quantidade de blocos no chão e aumenta a de inimigos levando, portanto, a tornar a fase menos trivial para se concluir) podem se tornar impossíveis de chegar ao fim. A função da equação (2.3) busca resolver este problema, minimizando mais a função quando um agente é capaz de concluir a fase completamente, e ainda adiciona maior dificuldade ao minimizar mais ainda a função utilizando o número de pulos.

O problema desta abordagem é o alto custo computacional de utilizar um agente para resolver as fases, ainda mais dentro de um algoritmo genético. Levando em conta 50 gerações e uma população de tamanho 28, para cada execução do algoritmo um agente jogaria 1400 níveis, ou seja, se o agente de Baumgarten for levado em conta, seriam 1400 execuções de um algoritmo A* por execução do algoritmo genético. Uma vez que isto gera um alto custo computacional (aproximadamente 105 minutos por execução, levando em conta o computador em que foram feitos os testes), não foi possível realizar execuções em alto volume como nos demais experimentos. Para este experimento, o conjunto de treinamento considerado, para fins de comparação justa, foi apenas o gerado pelo processamento do primeiro nível do jogo *Super Mario Bros*, o mesmo utilizado por (Volz et al., 2018). Para cada pedaço dos níveis foram realizadas 5 execuções do algoritmo NSGA-II e, ao invés de utilizar a técnica de *compromise programming* como critério de decisão entre as soluções não dominadas, todas as soluções não dominadas geradas foram observadas para uma maior variedade de possíveis soluções. A **Tabela 5.2** sintetiza os resultados, considerando as cinco partes das fases dividas em (Volz et al., 2018).

É possível observar que o desvio padrão é muito baixo ou arredondado para 0 nas três primeiras partes e que, com o uso de F_3 (ou seja, otimizando a função da equação (2.3)) juntamente com as funções das equações (2.1) ou (2.2) a grande maioria dos níveis gerados também são jogáveis, com algumas exceções quando estamos otimizando F_2, onde o número de inimigos é um fator que é aumentado. Isto pode mostrar beneficios do uso de um agente, porém este teste é de certa forma enviesado, já que o mesmo agente utilizado para testar as fases (o agente A* de Baumgarten) definiu se estas fases são jogáveis ou não. Para que a conclusão do

Parte do Nível	Média 1ª Fitness	Desvio Padrão (1ª Fitness)	Média F3	Desvio Padrão (F3)	Numero de Soluções	Soluções Jogáveis
Parte 1	0 (F_1 com $t = 1$)	0	-3.41	2.26	39	39
Parte 2	0 (F_1 com $t = 1$)	0	-4.39	2.42	46	46
Parte 3	0 (F_1 com $t = 0.7$)	0	-6.59	2.47	63	63
Parte 4	5.84 (F_2 com $t = 0.7$)	0.749	-2.65	1.98	38	35
Parte 5	5.5 (F_2 com $t = 0.7$)	0.63	-3.64	3.41	43	42

Tabela 5.2: Resultados dos experimentos multi-objetivo utilizando NSGA-II e as funções sugeridas em Volz et al. (2018)

uso do agente não seja enviesada, um agente diferente deveria ser utilizado para validar, porém este é um estudo para o futuro.

Como a menor quantia de partes jogáveis é 35 (da quarta parte) esta abordagem nos permite gerar 35 níveis jogáveis com progressão de dificuldade no eixo x. Porém, uma vez que estas 5 execuções por função otimizada levaram aproximadamente 20 horas, no momento parece não compensar em questão de tempo o uso do agente na função de fitness, mas sim como forma de desempate entre níveis gerados, mesmo que isto signifique perder a minimização de F_3, que se traduz na maximização de pulos (o que leva a fases mais interessantes, já que os pulos são assumidos como um indicador de dificuldade do nível por (Volz et al., 2018)).

A **figura 5.5** exemplifica uma fase jogável qualquer utilizando esta metodologia, juntando as cinco partes diferentes. Visualmente, é possível ver partes que o agente certamente precisaria executar a ação "pulo", assim como um pedaço do nível que não há chão, isto é, não há blocos na última linha da matriz que representa o nível. Após isso, o número de inimigos cresce consideravelmente, porém a quantidade de inimigos domina a sugestão de que deveriam também faltar pedaços do solo nestas partes. Volz et al. (2018) mostra um exemplo em que isto não acontece, porém o artigo não mostra muitos exemplos possivelmente à complexidade computacional de gerar estes níveis, assim como não otimiza F_3 simultaneamente.

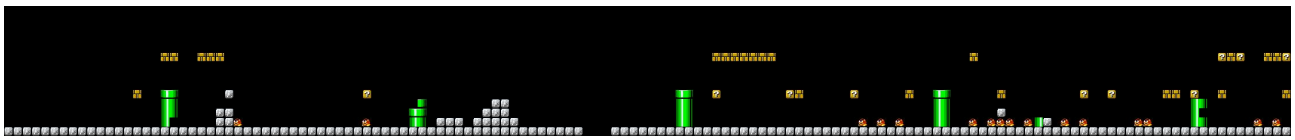


Figura 5.5: Nível qualquer juntando as cinco partes, conforme detalhado na seção 5.2.1

5.3 TOPOLOGIA DOS NÍVEIS

Assim como foram analisadas as funções propostas em (Volz et al., 2018), também foi observado o efeito da evolução de variáveis latentes aplicando nas entradas do modelo uma função diretamente ligada com a topologia das fases, no caso a função da equação (4.1), que busca maximizar um certo tipo de bloco, isto é, a quantia de vezes que um elemento (ou número, em termos literais) aparece na matriz que representa um nível. Para este experimento o bloco maximizado, ou seja, o valor de b na equação (4.1), foi 5 (inimigos) e 6 (canto superior esquerdo do cano).

Além da análise de um modelo treinado com o mesmo conjunto de (Volz et al., 2018), ou seja, apenas com a primeira fase do jogo *Super Mario Bros*, também foram realizados

treinamentos com cada um dos conjuntos presentes na **Tabela 4.1**, ou seja, com cada um dos conjuntos de fases disponíveis agrupados em mundo. Para cada conjunto foram realizadas 1000 iterações do PSO e do CMA-ES, sendo 500 para maximizar inimigos e 500 para maximizar canos, enquanto o NSGA-II realizou apenas 500 iterações, otimizando simultaneamente a função para $b = 5$ e $b = 6$. Adicionalmente, foram realizadas 500 iterações com entradas aleatórias por conjunto de treino, na intenção de comparar os efeitos do uso da LVE com entradas ruidosas sem cuidado algum. Em todos os casos, as entradas foram limitadas a números reais entre -2 e 2. Os resultados estão nas **tabelas 5.4** e **5.5**

Mundo	**Entrada**	**Média de Canos** ($\bar{x}$)	**Desvio Padrão (Canos)** (σ)	**Média de Inimigos** ($\bar{x}$)	**Desvio Padrão (Inimigos)** (σ)
Mundo 1	Aleatórias	0.482	0.715	1.77	1.57
	CMA-ES	**5.81**	0.558	1.17	0.522
	NSGA-II	1.44	0.807	**6.00**	1.85
	PSO	4.64	0.934	1.54	1.06
Mundo 2	Aleatórias	0.650	0.548	4.07	2.68
	CMA-ES	**3.43**	1.12	4.61	2.26
	NSGA-II	1.01	0.498	**9.27**	1.78
	PSO	3.25	0.882	4.74	2.32
Mundo 3	Aleatórias	0.280	0.512	2.29	1.83
	CMA-ES	**2.86**	0.345	3.91	1.51
	NSGA-II	1.51	0.686	**7.17**	1.83
	PSO	2.82	0.383	3.87	1.62
Mundo 4	Aleatórias	0.652	0.832	0.918	1.05
	CMA-ES	**5.68**	0.467	4.39	1.02
	NSGA-II	3.96	0.778	**5.19**	1.01
	PSO	5.31	0.640	4.40	1.01
Mundo 5	Aleatórias	0.230	0.527	3.05	2.14
	CMA-ES	**4.08**	0.599	3.41	1.38
	NSGA-II	0.530	0.809	**10.7**	2.21
	PSO	3.69	0.637	3.51	1.36
Mundo 6	Aleatórias	0.0900	0.293	0.0940	0.312
	CMA-ES	1.84	0.543	1.34	0.724
	NSGA-II	1.27	0.536	1.28	0.547
	PSO	**1.99**	0.0999	**1.57**	0.612
Mundo 7	Aleatórias	0.452	0.733	1.26	1.14
	CMA-ES	**4.84**	0.494	2.47	0.728
	NSGA-II	2.38	1.02	**4.50**	1.17
	PSO	4.62	0.705	2.50	0.758
Mundo 8	Aleatórias	0.778	0.922	3.56	1.96
	CMA-ES	**5.85**	0.398	3.91	0.870
	NSGA-II	1.64	1.01	**9.86**	2.05
	PSO	5.22	0.702	3.93	1.37

Tabela 5.3: Estatísticas dos conjuntos de treino maximizando o número de canos

As **tabelas 5.4** e **5.5** contém vários resultados que possivelmente são os mais interessantes do trabalho, já que demonstram o quanto a técnica de LVE altera o conteúdo gerado. O mais importante deles é que para qualquer método usado para a LVE - seja o CMA-ES, o PSO ou o NSGA-II - a função objetivo otimizada e, consequentemente, o tipo de elemento visado (maximizando canos na primeira tabela e inimigos na segunda) aparece em quantidade maior do que quando comparado com entradas aleatórias, significando que o uso da LVE possui sim impacto relevante pelo menos no que se diz respeito à topologia da fase. O CMA-ES e o PSO no mundo 1 maximizando canos, por exemplo, tem média de 5.81 e 4.64 canos respectivamente, contra uma média de apenas 0.482 canos levando em consideração entradas aleatórias. Outra discrepância grande é a do mundo 8 maximizando inimigos: enquanto o CMA-ES gera uma média de 16.4 inimigos e o PSO de 14.3, as entradas aleatórias tem média de 3.56 inimigos por nível gerado.

Além disso, na grande maioria das fases geradas a partir de múltiplos conjuntos de treinamento, o CMA-ES é o algoritmo que melhor converge, com exceção do mundo 6 para inimigos e canos. Para inimigos, por exemplo, o PSO tem média de 2.06, e o CMA-ES possui média de 1.86 inimigos por nível gerado. A diferença entre os valores da média das funções objetivo do PSO e o CMA-ES ainda é pequena e, caso algum teste estatístico corrobore que não são significativas, o PSO seria uma melhor alternativa ao CMA-ES devido ao tempo de execução menor, conforme abordado na sessão 5.1.

Em relação ao NSGA-II, embora ele não tenha superado no objetivo principal nenhum dos outros métodos de LVE, no oposto (número de inimigos quando se maximiza canos ou número de canos quando se maximizando canos) ele sempre é superior com exceção do mundo 6, o que poderia levar a um estudo da correlação entre o número de canos e de inimigos neste mundo específico. No mundo 3, por exemplo, a média de canos por nível gerado é de 1.51 e a de inimigos por nível gerado é de 7.17, enquanto as entradas aleatórias possuem média de 0.280 canos e 2.29 inimigos por nível gerado. Além disso, como já dito anteriormente, o NSGA-II ainda foi melhor que as entradas aleatórias, justificando seu uso numa situação de equilíbrio entre duas funções não correlacionadas em relação à topologia do nível. Outro detalhe que deve se ter atenção é que, para as análises do NSGA-II, foi utilizado um critério de desempate que equilibra as duas funções igualmente, e o algoritmo pode ou não ter gerado soluções não dominadas mais interessantes que os demais que a *compromise programming* não considerou.

5.3.1 Análise visual

Além de resultados quantitativos, o fato de o que estar sendo analisado ser as fases de um jogo permite analisar de forma visual os níveis gerados. Alguns casos interessantes foram selecionados. A **Figura 5.6** mostra 10 conteúdos gerados por uma GAN treinada com as amostras do mundo dois, de forma aleatória, algumas não sendo possível concluir, outras com alguma quantidade de canos e inimigos. De imediato, podemos afirmar que, visualmente, os níveis são diferentes porém com semelhanças reconhecíveis a olho nu. A **Figura 5.7** mostra conteúdos gerados pela mesma GAN, mas utilizando o CMA-ES como técnica de LVE e maximizando canos. Analisando estes níveis, podemos ver que a geração de canos foi de fato priorizada, com algumas ressalvas.

Nesta função, o bloco que corresponde ao número 6 da matriz faz contar como o elemento a ser maximizado, mas no conjunto de treinamento apresentado este elemento aparece uma única vez por agrupamento dos blocos 6, 7, 8 e 9, isto é, deve aparecer apenas uma vez por cano verde presente no nível. Em alguns dos níveis da **Figura 5.7** há canos sem o canto superior esquerdo, não contando para a função, assim como há canos quebrados que possuem mais de um bloco deste tipo, sendo que no treinamento cada cano possui apenas um desses blocos. Isto nos

leva a crer que o contexto não foi totalmente traduzido, sendo talvez ainda mais ignorado com métodos de LVE.

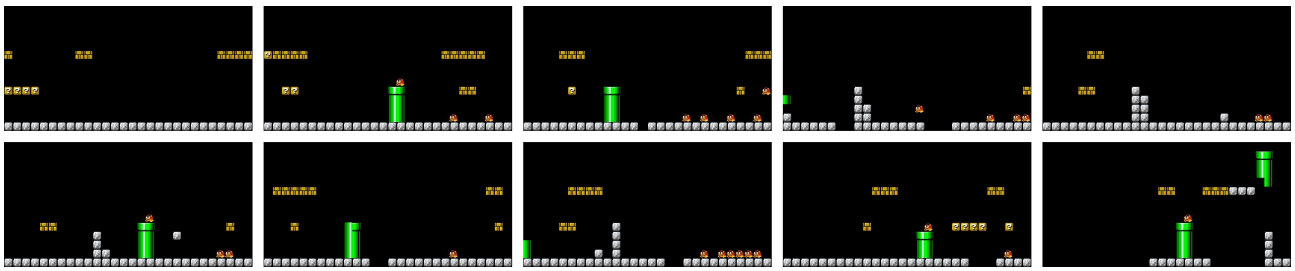


Figura 5.6: Fases aleatórias geradas pela GAN treinada com o mundo 2

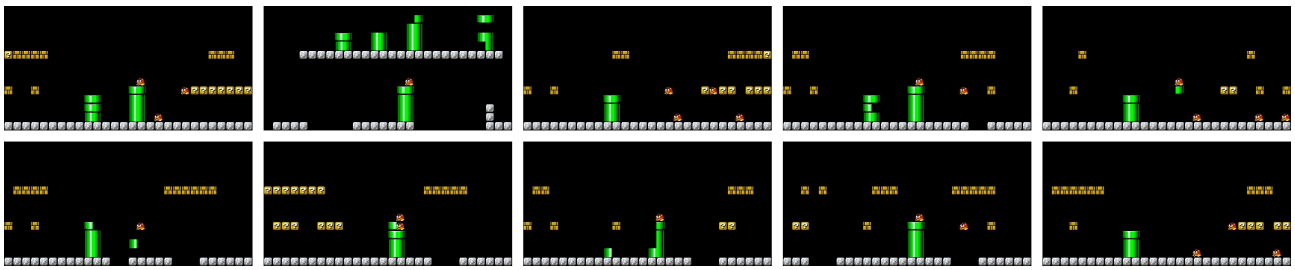


Figura 5.7: Fases geradas pela GAN treinada com o mundo 2 utilizando o CMA-ES como técnica de LVE, maximizando canos

Já num exemplo utilizando o mundo 5 como conjunto de treinamento, que tem fases mais planas, podemos ver como se comporta a função maximizando o número de inimigos. A **figura 5.8** mostra conteúdo aleatórios gerados pela GAN treinada com o mundo 5, já as **figuras 5.9** e **5.10** mostram níveis utilizando o CMA-ES e o PSO como técnicas de LVE, respectivamente, aplicando a equação (4.1) como função, maximizando o número de inimigos. Por fim, a **figura 5.11** mostra níveis gerados pelo NSGA-II. Visualmente, podemos afirmar que o PSO e o CMA-ES conduziram o espaço latente de forma a gerarem níveis razoavelmente parecidos. Já o NSGA-II conseguiu gerar alguns canos além dos inimigos, porém não em uma quantidade muito alta, já que não faz sentido gerar algo que você não vê tanto em seu conjunto de treinamento.

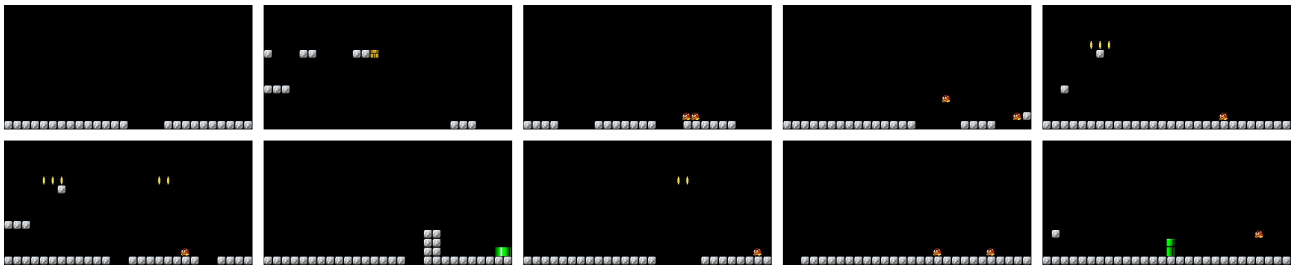


Figura 5.8: Fases aleatórias geradas pela GAN treinada com o mundo 5

Por fim, outro caso interessante é utilizar a LVE maximizando inimigos conhecendo o mundo 6 como conjunto de treino, conforme mostra a **figura 5.7**, em que a técnica de LVE utilizada foi o PSO. Neste caso, não há inimigos pois as fases do mundo 6 não possuem inimigos, são focadas em pulos complexos - e as simplificações do dataset da VGLC ignoram alguns outros elementos - mostrando que, se de fato não conhece aquilo, fica impossível gerar tal conteúdo.

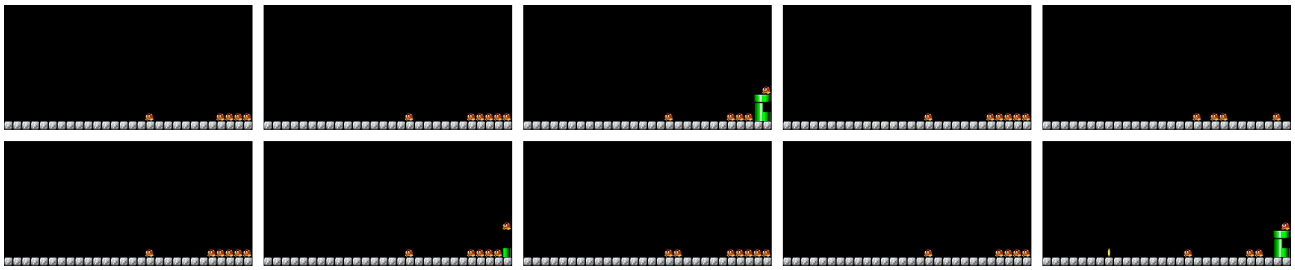


Figura 5.9: Fases geradas pela GAN treinada com o mundo 5 utilizando o CMA-ES como técnica de LVE, maximizando inimigos

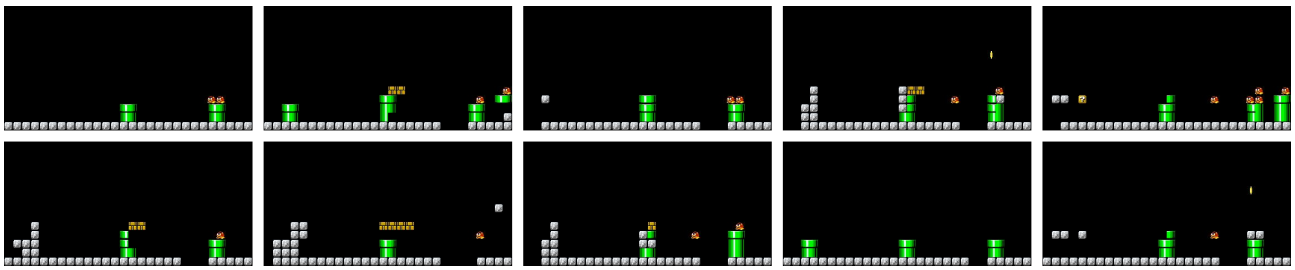


Figura 5.10: Fases geradas pela GAN treinada com o mundo 5 utilizando o PSO como técnica de LVE, maximizando inimigos

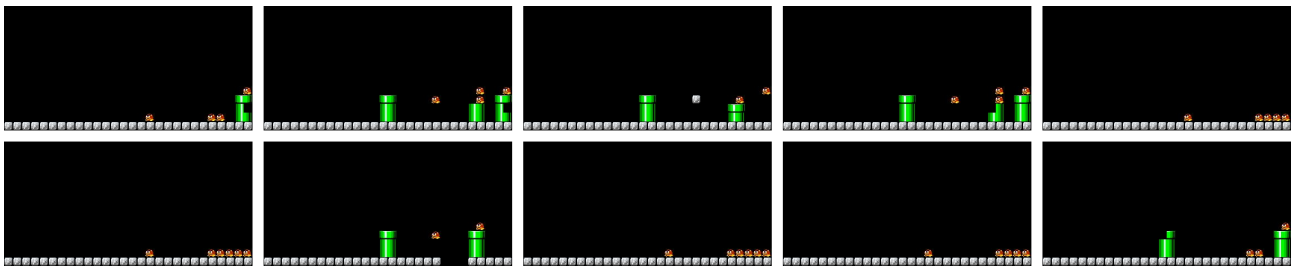


Figura 5.11: Fases geradas pela GAN treinada com o mundo 5 utilizando o NSGA-II como técnica de LVE, maximizando canos e inimigos simultâneamente

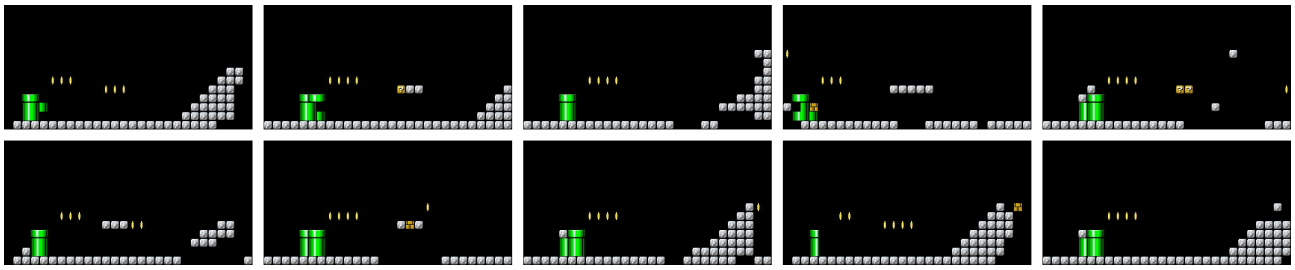


Figura 5.12: Fases geradas pela GAN treinada com o mundo 6 utilizando o CMA-ES como técnica de LVE, maximizando inimigos

5.4 CLASSIFICAÇÃO DOS PADRÕES GERADOS

Neste experimento, foi realizada uma tentativa de avaliar se o conteúdo gerado se assemelha suficientemente ou não com o conjunto de treino apresentado. Como um SVC de uma classe ou múltiplas não se mostraram suficientes para dividir as janelas deslizantes dos mundos, um modelo neural simples foi testado e, como mostrado no capítulo anterior, alcançou valores satisfatórios. Uma vez decidida a forma de classificar o padrão, esta rede foi utilizada para avaliar os pedaços de fase gerados, comparando as entradas geradas de forma aleatória e utilizando

o PSO e o CMA-ES como técnicas de evolução de variáveis latentes. Nesta comparação, nós utilizamos nove conjuntos diferentes de treinamento da rede geradora, sendo a primeira fase como em (Volz et al., 2018) e cada um dos oito mundos, totalizando 9 conjuntos de treino. Para cada conjunto, foram geradas 500 entradas aleatórias, 500 entradas com uso de CMA-ES como método de LVE e 500 com o uso do PSO, totalizando 1500 entradas por conjunto de treino, totalizando 1350 níveis avaliados. Neste experimento, a função otimizada foi a da equação (4.1) com $b = 6$, ou seja, maximizando o número de canos. Os gráficos presente nas figuras **5.13**, **5.14** e **5.15** sintetizam estes resultados, com uso de entradas aleatórias, entradas construídas pelo CMA-ES e PSO, respectivamente.

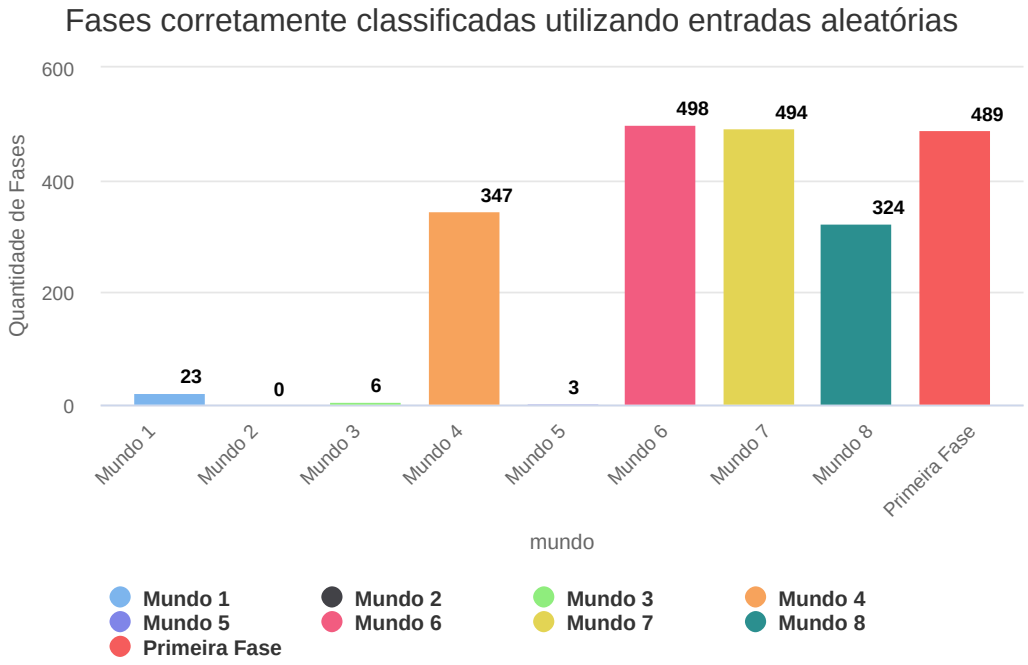


Figura 5.13: Classificação corretas das entradas aleatórias

Além desses gráficos, é interessante ressaltar que majoritariamente as fases foram classificadas dentro de um padrão específico pela GAN. A **Tabela 5.5** explicita os padrões de cada conjunto de treino, técnica e padrão.

Embora alguns mundos do primeiro ao quinto, com exceção do quarto, sejam classificados fora do padrão esperado, é importante ressaltar que eles não distoam na questão do padrão atribuído. A GAN treinada com o mundo 3, por exemplo, tem seus conteúdo gerados majoritariamente classificados como níveis do mundo 5, independente do tipo de entrada. Comparando as entradas aleatórias com técnicas de LVE, a afirmação de que o uso de uma técnica de manipulação de espaço latente não interfere no padrão que se encaixa um conteúdo gerado ganha um tanto de força. Porém, embora empiricamente provada com uma única função, esta possível conclusão só seria correta caso haja uma prova matemática para tal.

Esta sessão do capítulo ainda tenta justificar a escolha das redes treinadas com os mundos 1 e 6, que possuem mais níveis e são classificadas fora e dentro do padrão esperado, respectivamente, além da primeira fase (que é o mesmo conjunto de treinamento de (Volz et al., 2018)) para o experimento da sessão 5.2. Com estes três conjuntos de treinamento, é crível que se possa cobrir de forma satisfatória naturezas distintas de conjuntos de treino.

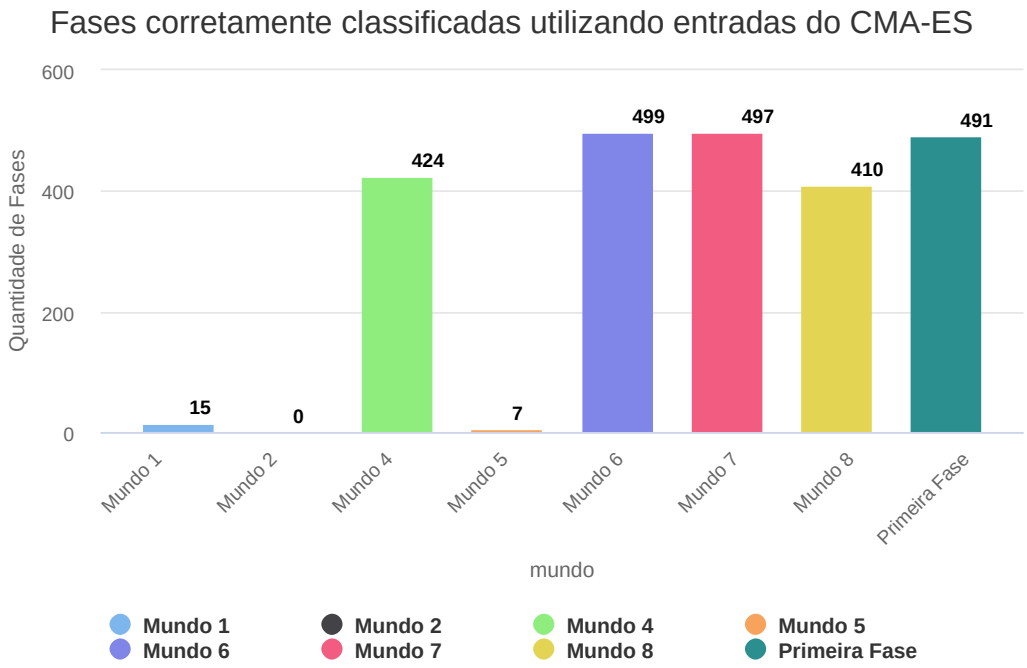


Figura 5.14: Classificação corretas das entradas do CMA-ES

5.5 JOGABILIDADE

Além das formas de análise já apresentadas, alguns dos conjuntos de níveis já apresentados foram avaliados utilizando o agente A* de Baumgarten. Das métricas fornecidas pelo agente, foram avaliadas a quantidade de níveis jogáveis, a porcentagem de conclusão dos níveis injogáveis (isto é, o quanto o agente se deslocou no eixo x da fase) e o número de ações que o algoritmo avaliou até chegar em uma solução (que pode ser tomada como um possível indicador de dificuldade). As **tabelas 5.6** e **5.7** mostram essas métricas para técnicas de LVE otimizando a função objetivo da equação (4.1) para $b = 6$ e $b = 5$, respectivamente, enquanto a **tabela 5.8** mostra estes resultados quando o NSGA-II minimiza ambas funções juntas.

Com exceção dos conjuntos de treinamento formado pelo processamento dos mundos 4, 7, e 8 não houve grande diferença na porcentagem dos níveis jogáveis dos 500 níveis gerados com cada conjunto de treino utilizando alguma técnica de LVE, seja com a função objetivo maximizando canos ou inimigos, com alguns casos até mesmo superando a porcentagem das entradas aleatórias. As ações também se mantiveram com valores razoavelmente parecidos. Possivelmente, a altura dos canos ou muitos inimigos fizeram o agente perder muitas vezes, algo que faz sentido levando em consideração o tipo de manipulação de espaço latente realizada.

De qualquer forma, isto mostra que para gerar níveis interessantes e que ainda sejam jogáveis utilizando técnicas de LVE um agente poderia auxiliar ou pelo menos ser utilizado como critério de desempate. Este experimento também mostra a importância do conjunto de treinamento e como ele influencia na forma que a rede geradora irá se comportar: levando em conta as entradas aleatórias e as diferenças em níveis jogáveis utilizando alguma técnica de LVE, os resultados foram diretamente influenciados pelo conjunto de treinamento do modelo. Os resultados são análogos a **tabela 5.8**, que mostra essas métricas para o NSGA-II utilizado como técnica de LVE, otimizando simultaneamente a função objetivo da equação (4.1) para canos e inimigos.

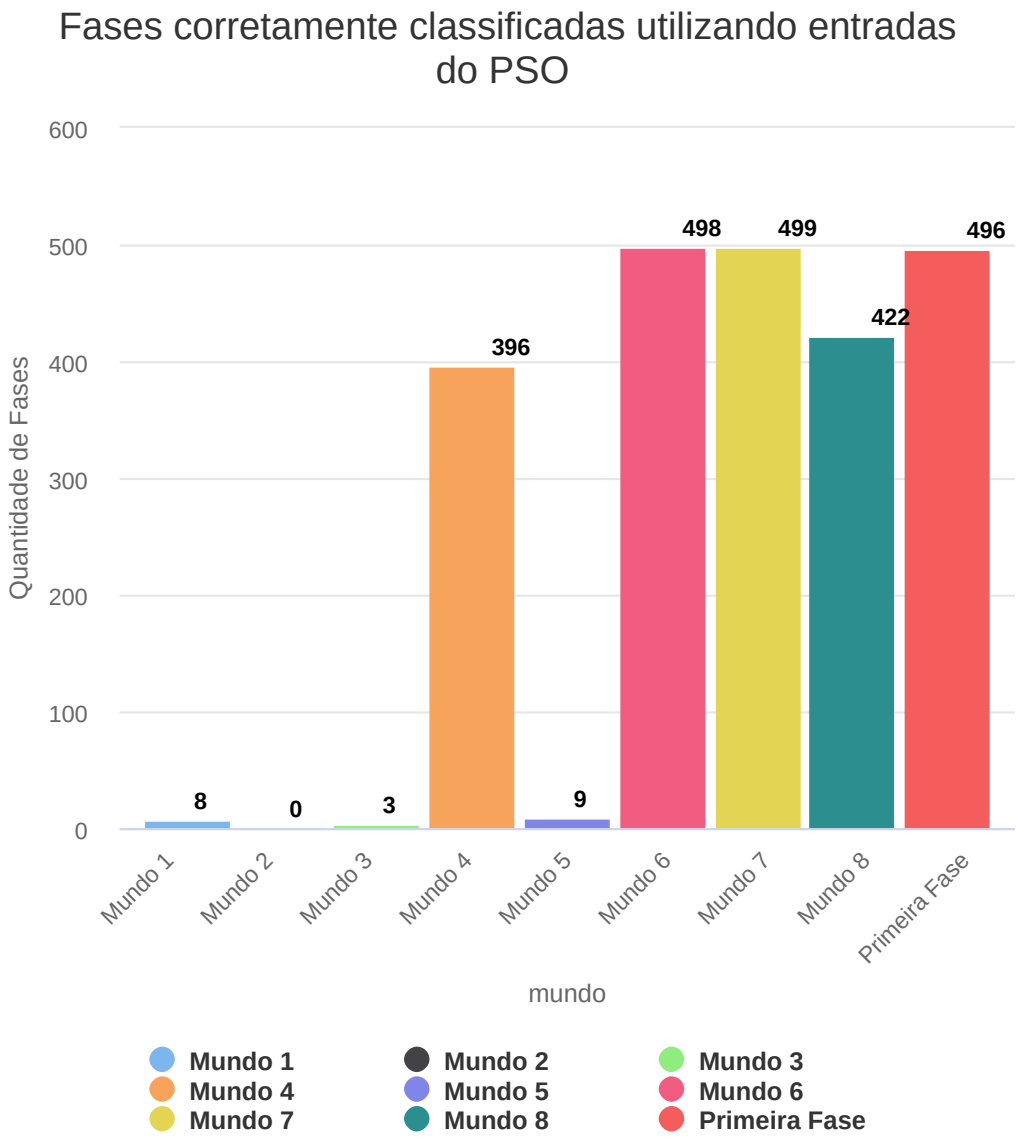


Figura 5.15: Classificação corretas das entradas do PSO

5.6 CONSIDERAÇÕES FINAIS

Este capítulo discutiu empiricamente como tanto o conjunto de treinamento quanto as funções objetivo utilizadas na técnica de LVE influenciam no conteúdo gerado de um gerador de fases do jogo *Super Mario Bros*. Para esta discussão, foram utilizadas tanto funções objetivo sugeridas em (Volz et al., 2018) quanto a função da equação (4.1). Com exemplos visuais e resultados quantitativos, foram comparados desempenho e valores finais dos algoritmos bioinspirados CMA-ES, PSO e NSGA-II em diversas situações com diversos conjuntos de treinamento, proporcionando uma ampla visão do uso do gerador de níveis proposto em (Volz et al., 2018) e como a técnica de LVE pode conduzir os níveis gerados a seguir certo padrão. No próximo capítulo, haverá uma sintese de tudo realizado neste trabalho, bem como discussão dos resultados e apontamento de trabalhos futuros que possam se desenvolver à partir deste.

Mundo	**Entrada**	**Média de Canos** ($\bar{x}$)	**Desvio Padrão (Canos)** (σ)	**Média de Inimigos** ($\bar{x}$)	**Desvio Padrão (Inimigos)**
Mundo 1	Aleatórias	0.482	0.715	1.77	1.57
	CMA-ES	0.140	0.391	**11.0**	0.79
	NSGA-II	**1.44**	0.807	6.00	1.85
	PSO	0.234	0.442	10.2	1.215
Mundo 2	Aleatórias	0.650	0.548	4.07	2.63
	CMA-ES	0.556	0.536	**15.3**	1.22
	NSGA-II	**1.01**	0.498	9.27	1.77
	PSO	0.712	0.574	13.8	1.14
Mundo 3	Aleatórias	0.280	0.512	2.29	1.830
	CMA-ES	0.806	0.519	**12.2**	1.03
	NSGA-II	**1.51**	0.686	7.17	1.82
	PSO	0.720	0.585	11.1	1.14
Mundo 4	Aleatórias	0.652	0.832	0.918	1.058
	CMA-ES	3.35	0.844	**7.00**	0.07
	NSGA-II	**3.96**	0.778	5.19	1.01
	PSO	2.97	1.07	6.80	0.449
Mundo 5	Aleatórias	0.230	0.527	3.05	2.14
	CMA-ES	0	0	**15.2**	1.16
	NSGA-II	**0.530**	0.809	10.7	2.21
	PSO	0.0380	0.270	13.5	1.52
Mundo 6	Aleatórias	0.0900	0.293	0.0940	0.31
	CMA-ES	1.27	0.673	1.86	0.62
	NSGA-II	1.27	0.536	1.28	0.54
	PSO	**1.38**	0.556	**2.06**	0.23
Mundo 7	Aleatórias	0.452	0.733	1.26	1.13
	CMA-ES	1.47	0.647	**6.78**	0.562
	NSGA-II	**2.38**	1.02	4.50	1.16
	PSO	1.14	0.697	6.43	0.63
Mundo 8	Aleatórias	0.778	0.922	3.56	1.96
	CMA-ES	1.14	0.917	**16.4**	0.928
	NSGA-II	**1.64**	1.01	9.86	2.05
	PSO	0.906	0.977	14.3	1.60

Tabela 5.4: Estatísticas dos conjuntos de treino maximizando o número de inimigos

Conjunto de treino	Entradas	Classe Predominante	Quantia Classificada
Primeira fase	Aleatória	1	489
	CMA-ES	1	491
	PSO	1	496
Mundo 1	Aleatória	7	454
	CMA-ES	7	460
	PSO	7	475
Mundo 2	Aleatória	8	382
	CMA-ES	8	412
	PSO	8	431
Mundo 3	Aleatória	5	456
	CMA-ES	5	489
	PSO	5	481
Mundo 4	Aleatória	4	347
	CMA-ES	4	424
	PSO	4	396
Mundo 5	Aleatória	7	464
	CMA-ES	7	472
	PSO	7	468
Mundo 6	Aleatória	6	498
	CMA-ES	6	499
	PSO	6	498
Mundo 7	Aleatória	7	494
	CMA-ES	7	497
	PSO	7	499
Mundo 8	Aleatória	8	324
	CMA-ES	8	410
	PSO	8	422

Tabela 5.5: Padrões da classificação de cada conjunto de treinamento

Conjunto	**Entradas**	**% de Soluções Jogáveis**	**Média de Conclusão de Níveis Injogáveis**	**Desvio Padrão (Conclusão Injogáveis)**	**Média Ações (Jogáveis)**	**Desvio Padrão (Ações dos Níveis Jogáveis)**
	Aleatórias	75.4	0.0829	0.0263	78.9	10.0
Mundo 1	CMA-ES	83.6	0.198	0.144	103.	40.0
	PSO	88.2	0.177	0.141	92.0	28.0
	Aleatórias	56.2	0.113	0.117	80.7	13.0
Mundo 2	CMA-ES	38.6	0.0820	0.0743	78.2	9.18
	PSO	49.4	0.0762	0.0669	78.1	8.03
	Aleatórias	67.4	0.156	0.140	81.1	18.4
Mundo 3	CMA-ES	93.6	0.0750	0.0265	80.9	9.77
	PSO	92	0.0835	0.0518	80.0	7.39
	Aleatórias	83.4	0.0863	0.0798	77.6	7.72
Mundo 4	CMA-ES	45.2	0.0941	0.0141	83.6	4.98
	PSO	30.2	0.0887	0.0223	82.9	6.10
	Aleatórias	67.2	0.0971	0.0687	74.3	6.52
Mundo 5	CMA-ES	78.4	0.0858	0.0127	73.5	4.28
	PSO	79.4	0.0882	0.0121	73.8	4.90
	Aleatórias	86.6	0.245	0.170	82.9	16.8
Mundo 6	CMA-ES	91	0.125	0.120	78.6	11.4
	PSO	94.4	0.0860	0.0714	77.4	7.25
	Aleatórias	64	0.0729	0.0322	75.9	5.66
Mundo 7	CMA-ES	10.2	0.0750	0.0263	78.2	3.06
	PSO	9.8	0.0713	0.0205	78.3	4.90
	Aleatórias	78.4	0.0845	0.0600	75.5	7.00
Mundo 8	CMA-ES	50.8	0.214	0.165	84.3	6.04
	PSO	50	0.211	0.162	86.0	7.63
	Aleatórias	68.4	0.112	0.106	79.6	16.8
1ª Fase	CMA-ES	55.8	0.0923	0.0695	80.1	16.4
	PSO	65.4	0.0942	0.0724	79.5	13.5

Tabela 5.6: Avaliação por agente com as técnicas de LVE otimizando a função (4.1), com $b = 6$, isto é, maximizando canos

Conjunto	Entradas	% de Soluções Jogáveis	Média de Conclusão de Níveis Injogáveis	Desvio Padrão (Conclusão Injogáveis)	Média Ações (Jogáveis)	Desvio Padrão (Ações dos Níveis Jogáveis)
Mundo 1	Aleatórias	75.4	0.0829	0.0263	78.9	10.0
	CMA-ES	63.2	0.0640	0.0176	82.6	8.32
	PSO	48.4	0.0669	0.0354	82.4	10.4
Mundo 2	Aleatórias	56.2	0.113	0.117	80.7	13.0
	CMA-ES	57	0.0755	0.0668	80.4	8.21
	PSO	60.4	0.0714	0.0652	80.3	9.96
Mundo 3	Aleatórias	67.4	0.156	0.140	81.1	18.4
	CMA-ES	65.8	0.0659	0.0185	76.3	6.88
	PSO	76	0.0695	0.0287	77.9	15.1
Mundo 4	Aleatórias	83.4	0.0863	0.0798	77.6	7.72
	CMA-ES	4	0.0795	0.00670	84.7	14.2
	PSO	17.2	0.0790	0.0192	77.4	5.70
Mundo 5	Aleatórias	67.2	0.0971	0.0687	74.3	6.52
	CMA-ES	99.8	0.102	0.102	71.4	0.931
	PSO	93.6	0.164	0.122	71.3	0.585
Mundo 6	Aleatórias	86.6	0.245	0.170	82.9	16.8
	CMA-ES	93.2	0.136	0.118	78.5	10.5
	PSO	93.6	0.113	0.0888	78.0	7.58
Mundo 7	Aleatórias	64	0.0729	0.0322	75.9	5.66
	CMA-ES	93.8	0.145	0.142	82.1	9.37
	PSO	92.6	0.130	0.146	78.5	10.3
Mundo 8	Aleatórias	78.4	0.0845	0.0600	75.5	7.00
	CMA-ES	75.2	0.0959	0.0328	76.2	6.87
	PSO	81.8	0.0889	0.0167	74.3	6.43
1ª Fase	Aleatórias	68.4	0.112	0.106	79.6	16.8
	CMA-ES	88.2	0.0899	0.0611	83.8	10.4
	PSO	91.4	0.0926	0.0797	83.3	10.8

Tabela 5.7: Avaliação por agente com as técnicas de LVE otimizando a função (4.1), com $b = 5$, isto é, maximizando inimigos

Conjunto	**% de Soluções Jogáveis**	**Média de Conclusão de Níveis Injogáveis**	**Desvio Padrão (Conclusão Injogáveis)**	**Média Ações (Jogáveis)**	**Desvio Padrão (Ações dos Níveis Jogáveis)**
Mundo 1	0.608	0.0805	0.0264	84.1	11.4
Mundo 2	0.412	0.0807	0.0883	83.1	10.9
Mundo 3	0.778	0.0650	0.0361	76.7	5.68
Mundo 4	0.18	0.0851	0.0113	83.6	4.14
Mundo 5	0.894	0.0917	0.00355	71.9	2.76
Mundo 6	0.92	0.117	0.0697	78.1	7.85
Mundo 7	0.486	0.0813	0.0637	82.7	16.2
Mundo 8	0.564	0.0956	0.0590	78.1	7.32
Primeira Fase	0.92	0.104	0.103	83.3	11.4

Tabela 5.8: Avaliação por agente com o NSGA-II otimizando a função (4.1), com $b = 5$ e $b = 6$ simultaneamente

6 CONCLUSÃO

Este capítulo concluí o estudo sumarisando seus diversos experimentos e resultados alcançados. Ele também sugere possíveis aplicações dos modelos e técnicas de LVE utilizadas no trabalho realizado. Finalizando, são apresentados estudos em PCGML que podem ser derivados deste trabalho.

6.1 DISCUSSÃO DOS RESULTADOS

A GAN utilizada em (Volz et al., 2018) é pública, assim como o agente A* apresentado em (Togelius et al., 2010) e os níveis utilizados nos conjuntos de treinamento da VLGC (Summerville et al., 2016). O objetivo deste estudo foi avaliar o impacto da técnica de LVE utilizando três algoritmos bioinspirados diferentes: o CMA-ES, o PSO, e também o NSGA-II para experimentos multi-objetivo conforme Volz et al. (2018) sugere. Além de seguir a sugestão de estudo futuro de Volz et al. (2018) utilizando um algoritmo multi-objetivo na técnica de LVE (no caso o NSGA-II), também foram realizadas comparações dos algoritmos bioinspirados em desempenho e valores atingidos tanto nas funções propostas por Volz et al. (2018) quanto na função objetivo da equação (4.1) com conjuntos de treinamento de granularidade maior do que em (Volz et al., 2018). Por fim, houve uma preocupação em avaliar jogabilidade e qualidade das soluções, avaliações estas realizadas através de métricas extraidas pelo agente A* de Baumgarten e também por um classificador neural, na tentativa de checar se as soluções geradas seguiam um padrão e se a LVE alteraria este padrão, situação que não aconteceu.

Realizando estes diversos experimentos, é possível ilustrar como a técnica de LVE pode ser usada para que os conteúdos do gerador sigam alguma condição imposta, condição esta que se traduz numa função objetivo. Porém, sem o devido cuidado ou filtro, níveis que não podem ser completados podem ser gerados. Também foram feitos experimentos que fortalecem a hipótese de que o uso da LVE não altera suficientemente o padrão dos conteúdos gerados à ponto de se tornarem fora do padrão do modelo neural, e as evidências visuais levam a acreditar que o modelo gera níveis suficientemente diferentes com uso ou não da técnica de LVE.

O uso do modelo explorado treinado com diferentes conjuntos e utilizando funções-objetivo diversas na técnica de LVE gera níveis do jogo *Super Mario Bros* diferentes entre si, e um agente pode ser utilizado para decidir a jogabilidade destes níveis, bem como utilizado dentro da própria LVE se for de execução menos custosa do que o agente A* proposto em (Togelius et al., 2010).

6.2 POSSÍVEIS APLICAÇÕES

O modelo proposto pode ser utilizado para gerar níveis jogáveis e interessantes do jogo *Super Mario Bros*, com atributos diversos a depender das funções objetivo utilizadas durante a realização da LVE. O modelo, treinado em diversos conjuntos distintos, poderia gerar níveis que seriam filtrados no quesito jogabilidade pelo agente A* de Baumgarten, para então serem apresentados a um jogador humano. Uma aplicação deste tipo tem a possibilidade de tornar um jogo com um fator *replay* infinito, em que cada nível seria gerado proceduralmente seguindo alguns padrões humanos estabelecidos previamente no conjunto de treinamento.

6.3 ESTUDOS FUTUROS

Este estudo considera apenas uma única arquitetura e um único jogo. Volz et al. (2018) sugere que o modelo proposto é genérico o suficiente para generalizar qualquer jogo que possa ser representado no formato proposto pelo repositório aberto da *Video Game Level Corpus*, hipótese esta que poderia ser testada empiricamente. Há também muitas arquiteturas propostas para PCGML, como a CESAGAN proposta em Torrado et al. (2019) e a MultiGAN proposta em Capps e Schrum (2021), que poderiam ser comparadas com a arquitetura da MarioGAN tanto no jogo *Super Mario Bros* quanto em outros, como o *The Legend Of Zelda* utilizado em (Torrado et al., 2019).

Além disso, todos os modelos citados para comparação utilizam camadas neurais convolucionais, e não *Vision Transformers*, arquitetura avançada de aprendizado profundo que vem ganhando popularidade nos ultimos anos. Lee et al. (2021) propõe o uso de *Vision Transformers* dentro de GANs, mas não na área de PCGML, sendo também uma possibilidade de estudo.

As funções objetivo também podem variar. Neste trabalho, apenas quatro foram comparadas e executadas, mas qualquer função objetivo que faça sentido pode ser inclusa dentro de algum algoritmo de LVE sem grandes complicações além de sua implementação, com o único limitante sendo a criatividade de quem as elabora.

Por fim, já que a VGLC representa os níveis em forma de texto, *Large language models*(LLM) também poderiam ser utilizados para gerar estes níveis. Sudhakaran et al. (2023) propõe o uso de LLMs para PCGML utilizando o modelo GPT-2 como base com auxílio de *novelty search* e operadores de mutação customizados, chamando o modelo de MarioGPT. Um estudo poderia ser desenvolvido em cima do modelo MarioGPT, comparando com o modelo explorado neste trabalho (a MarioGAN). Um fato interessante é que Sudhakaran et al. (2023) também utiliza o agente A* de Baumgarten para teste de jogabilidade dos níveis, o que já estabelece algo em comum entre os dois estudos.

REFERÊNCIAS

Arjovsky, M., Chintala, S. e Bottou, L. (2017). Wasserstein gan.

Blank, J. e Deb, K. (2020). pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509.

Bontrager, P., Togelius, J. e Memon, N. (2017). Deepmasterprint: Generating fingerprints for presentation attacks.

Capps, B. e Schrum, J. (2021). Using multiple generative adversarial networks to build better-connected levels for mega man.

Deb, K. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. Em IEEE, editor, *IEEE Transactions on evolutionary computation, Vol. 6*.

Fadaeddini, A., Majidi, B. e Eshghi, M. (2018). A case study of generative adversarial networks for procedural synthesis of original textures in video games. Em *2018 2nd National and 1st International Digital Games Research Conference: Trends, Technologies, and Applications (DGRC)*, páginas 118–122.

Ferreira, L. N., Lelis, L. H. S. e Whitehead, J. (2020). Computer-generated music for tabletop role-playing games.

Giacomello, E., Lanzi, P. L. e Loiacono, D. (2018). Doom level generation using generative adversarial networks.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. e Bengio, Y. (2014). Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3.

Hansen, N., Akimoto, Y. e Baudis, P. (2019). CMA-ES/pycma on Github. Zenodo, DOI:10.5281/zenodo.2559634.

Hansen, N., Müller, S. D. e Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1):1–18.

Hendrikx, M., Meijer, S., Van Der Velden, J. e Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Trans. Multimedia Comput. Commun. Appl.*, 9(1).

Hong, S., Kim, S. e and, S. K. (2019). Game sprite generator using a multi discriminator gan. *KSII Transactions on Internet and Information Systems*, 13(8):4255–4269.

Jin, Y., Zhang, J., Li, M., Tian, Y., Zhu, H. e Fang, Z. (2017). Towards the automatic anime characters creation with generative adversarial networks.

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. Em *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, páginas 1942–1948 vol.4.

Lee, K., Chang, H., Jiang, L., Zhang, H., Tu, Z. e Liu, C. (2021). Vitgan: Training gans with vision transformers.

Liu, J., Snodgrass, S., Khalifa, A., Risi, S., Yannakakis, G. N. e Togelius, J. (2020). Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1):19–37.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. e Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. Em *Advances in Neural Information Processing Systems 32*, páginas 8024–8035. Curran Associates, Inc.

Rao, S. (2010). A multiobjective genetic algorithm for feature selection in data mining.

Rebouças Serpa, Y. e Formico Rodrigues, M. A. (2019). Towards machine-learning assisted asset generation for games: A study on pixel art sprite sheets. Em *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, páginas 182–191.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. e Lee, H. (2016). Generative adversarial text to image synthesis.

Ringuest, J. L. (1992). *Compromise Programming*, páginas 51–59. Springer US, Boston, MA.

Srinivas, N. e Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248.

Sudhakaran, S., González-Duque, M., Glanois, C., Freiberger, M., Najarro, E. e Risi, S. (2023). Mariogpt: Open-ended text2level generation through large language models.

Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A. K., Isaksen, A., Nealen, A. e Togelius, J. (2018). Procedural content generation via machine learning (pcgml).

Summerville, A. J., Snodgrass, S., Mateas, M. e n'on Villar, S. O. (2016). The vglc: The video game level corpus. *Proceedings of the 7th Workshop on Procedural Content Generation*.

Togelius, J., Karakovskiy, S. e Baumgarten, R. (2010). The 2009 mario ai competition. páginas 1 – 8.

Togelius, J., Yannakakis, G. N., Stanley, K. O. e Browne, C. (2011). Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186.

Torrado, R. R., Khalifa, A., Green, M. C., Justesen, N., Risi, S. e Togelius, J. (2019). Bootstrapping conditional gans for video game level generation.

Volz, V., Schrum, J., Liu, J., Lucas, S. M., Smith, A. M. e Risi, S. (2018). Evolving mario levels in the latent space of a deep convolutional generative adversarial network. Em *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2018)*, New York, NY, USA. ACM.

Zhu, J.-Y., Park, T., Isola, P. e Efros, A. A. (2020). Unpaired image-to-image translation using cycle-consistent adversarial networks.