UNIVERSIDADE FEDERAL DE UBERLÂNDIA
CAMPUS SANTA MÔNICA
FACULDADE DE ENGENHARIA ELÉTRICA

Pedro Arantes Mendonça Toledo Almeida

TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL PARA GERAÇÃO DE INSUMOS 2D

UBERLÂNDIA
2019

PEDRO ARANTES MENDONÇA TOLEDO ALMEIDA

TÉCNICAS DE INTELIGÊNCIA ARTIFICIAL PARA GERAÇÃO DE INSUMOS 2D

Trabalho de Conclusão de Curso submetido à Universidade Federal de Uberlândia, como requisito necessário para obtenção do grau de Bacharel em Engenharia de Computação

Uberlândia, Julho de 2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

PEDRO ARANTES MENDONÇA TOLEDO ALMEIDA

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Engenharia de Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Prof. Dr. Keiji Yamanaka
Universidade Federal de Uberlândia - UFU

Prof. Dr. Alexandre Cardoso
Universidade Federal de Uberlândia - UFU

Prof. Dr. Igor Santos Peretta
Universidade Federal de Uberlândia - UFU

Uberlândia, XX de Julho de 2019

Agradecimentos

A todos que de forma direta ou indireta pavimentaram o caminho para que hoje estivesse aqui.

Resumo

O cenário atual da produção artística para jogos e animações passou por uma transformação profunda com a revolução da computação gráfica da segunda metade da década de 90. A presença ubíqua de animações feitas à mão nos cinemas e jogos com *sprites* também desenhados manualmente deram lugar a grandes produções que fazem o uso extenso, ou em muitos casos exclusivo, de imagens 3D geradas por computador e jogos em 2D que passaram a atender apenas um pequeno nicho devido a elevados custos e menor consistência nos insumos em 2D, que não acompanhou os avanços feitos em 3D. O trabalho apresenta a elaboração e implementação de duas técnicas que visam auxiliar artistas e designers no desenvolvimento de insumos em 2D, sendo elas o uso de Redes Generativas Antagônicas para a geração de *sprites*, e algoritmo genético para a geração automatizada de níveis de jogos 2D, a partir de abstrações feitas sobre princípios de design de jogos e análise de grafos. Embora geração de novos *sprites* obteve sucesso moderado devido aos altos custos computacionais, a implementação de algoritmo genético se provou extremamente eficiente em auxiliar no processo de criação de níveis. A partir do trabalho realizado é possível estimar que com os avanços constantes em disponibilidade de recursos computacionais, o uso de modelos generativos, assim como o de algoritmos genéticos serão cada vez mais presentes na caixa de ferramentas de artistas e designers.

Palavras-chave: Modelos Generativos, Rede Generativa Antagônica, Computação Gráfica, Ingeligência Artificial, Algoritmo Genético.

Abstract

The current artistic landscape in the production of video-games and animation went through radical change with the computer graphics revolution before the turn of the millennium. The ubiquitous presence of hand drawn animation in the theaters and games made with, also hand drawn, sprites gave place to production that made extensive, and sometimes exclusive, of 3D computer generated imagery and 2D games that now find themselves limited to a small niche due to the elevated costs and reduced consistency in 2D resources, since the tools for making them didn't evolve nearly as much as their 3D counterparts. This work then presents the development and implementation of two techniques aimed to help artists and designers in the development of 2D resources, specifically through the use of Generative Adversarial Networks for the generation of sprites and through the use of Genetic Algorithms to help automate level design. Despite the moderate success obtained in the sprite generation, due to the limitations of computational power, the implementation of genetic algorithm proved itself to be extremely efficient in helping the level design process. Through the work done it becomes possible to estimate that the constant advances in the availability of computational resources will make the use of Generative models as well as Genetic algorithms for 2D asset creation a reliable tool in the 2D designer and artist's toolkit.

Keywords: Artificial Intelligence, Computer Graphics, Generative Adversarial Networks, Genetic Algorithms.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Interface de programação de aplicações
POO	Programação Orientada a Objetos
RNN	Rede Neural Recorrente

Sumário

1 Introdução

A história das artes visuais é quase tão antiga quanto a humanidade em si, existindo antes mesmo da linguagem escrita [Tversky 2014, Kantrowitz, Brew e Fava 2011]. No início, os pictogramas eram uma forma de conhecimento especializado voltado a representar objetos e conceitos abstratos, mas propósito explicitamente artístico já existia em desenhos e pinturas com pigmentação primitiva quarenta mil anos atrás [Kleiner 2016]. Ao longo do tempo, esse processo passou por uma evolução constante, tendo com especial relevância o surgimento de técnicas de animação, mais de cinco mil e duzentos anos atrás no Irã [Cohn 2006].

Durante o processo milenar da história da arte, inúmeras evoluções foram realizadas em representações e abstrações de ideias e objetos no plano [Honour e Fleming 2005]. Diversas novas formas de mídia como as animações tradicionais e os jogos eletrônicos se tornaram enormes meios através dos quais a arte visual se tornava algo mais acessível para o público de massa, difundindo-se e tornando-se algo do cotidiano das pessoas.

Juntamente a esses avanços, é extremamente importante notar que algo ocorria em paralelo. O avanço da computação gráfica trouxe enormes mudanças para a forma na qual as coisas eram não somente vistas, mas criadas [Foley et al. 1996], tanto em 2D quanto em 3D. Até meados da década de 80, a computação gráfica, em particular sua aplicabilidade em 3D era um campo restrito e extremamente especializado. Porém, com avanços regulares e frequentes a viabilidade da utilização do 3D era cada vez maior em todos os setores. Filmes inteiros em computação gráfica 3D tiveram seu início com *Toy Story* e o filme brasileiro Cassiopeia na década de 90, e hoje em dia já são ubíquos no cinema, e as aplicações de 3D ainda na mesma década passaram para aplicações em tempo real, pois ainda na mesma década *Quake* se tornaria o primeiro jogo eletrônico a ser totalmente em 3D.

As ferramentas disponibilizadas aos artistas 3D, sejam eles modeladores, escultores ou animadores, evoluiu a uma taxa explosiva, e essa evolução ainda não parou. O surgimento de novas tecnologias como aplicações de cinemática inversa e fotogrametria [Sumner et al. 2005, Mikhail, Bethel e McGlone 2001] tornaram o trabalho de criar e dar vida a objetos em 3D uma tarefa mais atrativa para artistas, designers de jogos e produtores de filmes de animação no mundo todo.

Apesar das evoluções significativas na arte digital em 2D, as vantagens oferecidas por aplicações em 3D se tornavam cada vez mais claras. O fato de um mesmo modelo 3D poder ser rotacionado, transladado, alterado em escala em vários eixos e visto a partir de uma câmera no espaço sendo então projetado no plano da tela permite que um mesmo

modelo seja visto de inúmeras diferentes maneiras, enquanto na arte em 2D, para que um mesmo objeto seja visualizado de outra forma o mesmo precisa ser refeito.

O resultado disso foi de tornar cada vez menos competitiva a produção de arte digital em 2D, se tornando mais marginalizadas com apenas um filme em nível global [Lester 2010] em 2009. Nos campo dos jogos o processo foi ainda mais acelerado, com os jogos em 2D perdendo espaço antes mesmo da chegada do *PlayStation 2*, *GameCube* e *Xbox*, e isso ocorreu apesar do fato de que o hardware anterior não era sequer totalmente capaz de fazer operações hoje consideradas muito simples em 3D como correção por perspectiva [Hecker 1996]. Estes fatores levam a indicar que o artista 2D do século XXI está carente da riqueza de ferramentas presente na sua contrapartida tridimensional.

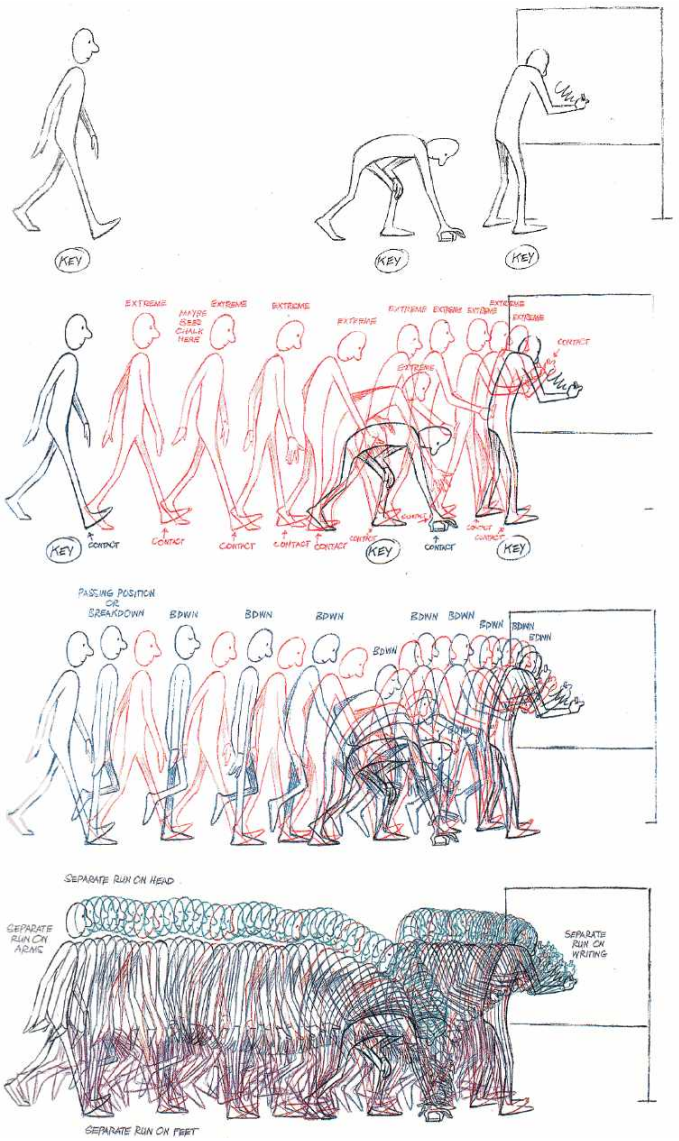
1.1 Problemática

A criação de recursos em 2D é um campo extremamente variado, na produção de um jogo eletrônico apenas, esse processo seria fundamental nas etapas de arte conceitual, *storyboarding*, criação de texturas (ou *sprites*), animação, design de níveis entre outros. Como as técnicas e problemas enfrentados são diversos, serão analisados com maior atenção aqueles que serão tratados durante o trabalho, sendo esses a geração de *sprites* e design de níveis.

A arte digital em 2D e 3D, embora partilhem de vários princípios, tem algumas diferenças fundamentais. A mais importante dessas diferenças é que cada desenho feito em um canvas (seja *pixel art* ou arte vetorizada) representa nada mais que uma abstração do artista, isto é, uma instância de algo projetado sobre a tela de uma forma direta, enquanto no 3D, através da lógica de espaço virtual, o objeto será projetado na tela a partir de uma câmera, e portanto o mesmo objeto pode se encontrar em inúmeros estados. Este fato permite ao artista e designer 3D fazer o uso extensivo de um mesmo modelo, muitas vezes tendo um único modelo 3D para todas as vezes e formas que o mesmo irá aparecer no produto, enquanto o artista em 2D, como fica extremamente evidente na Figura 1, durante o processo de criação em 2D envolve muitas instâncias do mesmo objeto sendo representado, pois todo o trabalho de câmera e situação da instância do objeto está diretamente nas mãos do artista, o que pode ser um volume muito grande de quadros para uma animação extremamente simples [Kuperberg 2012].

Jogos eletrônicos como *Persona 4 Arena* possuem, somente para a animação de personagens, mais de 700 *sprites* de animação por personagem, o que evidencia o volume de trabalho necessário para que seja possível fazer animações fluidas e agradáveis aos olhos de quem a vê. Portanto seria crucial a um artista poder ter ferramentas que a partir de um volume de trabalho já feito, é capaz de complementar o seu trabalho de ter que criar, desenhar e colorir cada quadro de animação.

Figura 1 – Rascunho geral de uma animação



Fonte: WILLIAMS, 2012[1].

Para o designer de níveis o desafio é mais abstrato, o mesmo deve ser capaz de abstrair uma organização de elementos virtuais que, quando convertidos em objetos do jogo terão o objetivo de alcançar ao menos um dos temas de design de níveis, sendo eles:

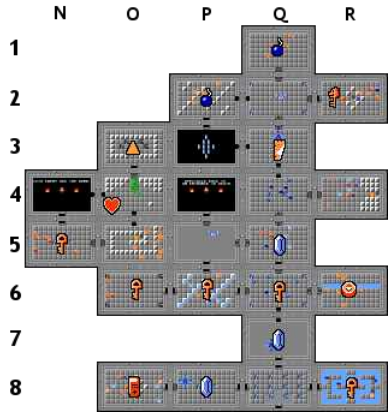
1. Sobreviver/Escapar
2. Explorar
3. Educar

Por ser uma atividade complexa, e extremamente subjetiva, as tentativas de automatizar esse processo costumam ter uma abordagem bottom-up no caso de geração

[1] WILLIAMS, Richard. The animator's survival kit: a manual of methods, principles and formulas for classical, computer, games, stop motion and internet animators. Macmillan, 2012.

procedural [Shaker, Togelius e Nelson 2016]. As tentativas de uma abordagem top-down feitas através de buscas costumam ter restrições simples e resultados ineficazes [Togelius et al. 2011]. Ao analisar um nível de um *game*, mesmo que simples, como *The Legend of Zelda* (1986) mostrado na Figura 2, é possível ver que a quantia de maneiras de se organizar salas com até 4 conexões, com vários conteúdos diferentes possíveis seria fazer uma busca em uma população de soluções que tende rapidamente ao infinito, portanto métodos analíticos de geração automática de design de níveis não foram bem sucedidos no passado.

Figura 2 – *Dungeon* em *The Legend of Zelda*



Fonte: Imagem montada pelo autor[1].

Para os dois problemas enfrentados por artistas e designers, soluções interessantes ainda não exploradas podem ser encontradas em novos modelos generativos [Wang e Gupta 2016] ou em aplicações inovadoras de algoritmos genéticos [Gandomi, Alavi e Ryan 2015].

1.2 Objetivo

1.2.1 Objetivo Geral

O objetivo consiste em desenvolver um modelo generativo que será criado a partir de uma varredura do estado da arte, visando a geração de *sprites* e design de níveis de forma automata com a combinação de diferentes métodos.

1.2.2 Objetivos Específicos

Considerando o trabalho a ser feito e o objetivo apresentado, tem-se os seguintes objetivos específicos:

- Pesquisar o estado da arte em Algoritmos Genéticos aplicados em geração e analise de grafos

[1] Nintendo, *The Legend of Zelda*. Famicom, 1982.

- Pesquisar o estado da arte em arquiteturas de Redes Neurais Artificiais generativas
- Modelar e desenvolver os algoritmos necessários para a aplicação
- Implementar os algoritmos desenvolvidos no motor de jogos Unity3D, de forma a transformar os algoritmos em ferramentas
- Realizar testes e melhoras iterativas nos modelos desenvolvidos
- Analisar e publicar os resultados obtidos

1.3 Organização

O trabalho está organizado na seguinte forma:

Capítulo 2: os conceitos relacionados a Algoritmos Genéticos serão abordados, e como aplicar o mesmo na criação de grafos conectados que irão representar um nível de um *game*.

Capítulo 3: a pesquisa relacionada a Modelos Generativos, tal como suas aplicações, diferentes implementações, arquiteturas, modelos de aprendizagem e modelagens matemáticas.

Capítulo 4: o desenvolvimento da implementação será detalhada, tal como as exatas especificações e detalhamento do funcionamento do algoritmo genético e modelo generativo desenvolvido especificadamente para a criação das ferramentas desejadas.

Capítulo 5: os resultados obtidos, incluindo as melhorias realizadas ao longo do processo iterativo de criação, modificação e treinamento de sistemas de inteligência artificial.

2 *Grafos, Design de Níveis e Algoritmos Genéticos*

2.1 Definições

O capítulo irá conter uma abordagem interdisciplinar a criação de níveis, portanto é crucial entender os conceitos que estarão interagindo uns com os outros. Pois ao final do capítulo é necessário compreender a relação entre design de níveis e grafos, e como seria possível aplicar os algoritmos genéticos a esses grafos.

2.1.1 Grafos

A palavra grafo, como é usada atualmente matemática e computação, foi originalmente usado por James Joseph Sylvester no final do século XIX [Sylvester 1878, Sylvester 1878]. Desde então, o campo da teoria dos grafos se tornou parte fundamental da matemática discreta e computação.

O grafo trata-se da estrutura representativa de um conjunto de objetos nos quais pares de objetos estão "relacionados"de alguma forma. Os objetos correspondem a abstrações matemáticas chamadas de vértices (também chamados de nós ou pontos) e cada um dos pares relacionados de vértices é chamado de aresta (também chamado de ligação ou linha) [Gross e Yellen 2004].

As arestas dos grafos podem ser direcionadas ou não direcionadas. Para melhor ilustrar esses conceitos, podemos imaginar um grafo que representa uma família (como a árvore genealógica). Nesse grafo, relações genéricas de parentesco seriam não-direcionadas, pois A é parente de B se e somente se B é parente de A. Já para relações específicas de parentesco, podemos ter relações que são direcionadas, como por exemplo A ser pai de B, e B ser filho de A.

2.1.2 Grafos Conectados Não-Direcionados

Para o escopo do projeto, interessa somente um subconjunto específico de grafos, no caso os grafos **conectados** e **não-direcionados**. Um grafo não-direcionado será chamado de conectado quando para qualquer par de vértices, há um caminho para se chegar de um para o outro. Em outras palavras, em um grafo conectado não há vértices inalcançáveis.

Para o estudo de conectividade dos grafos, uma ferramenta fundamental é o teorema de Menger, provado por Karl Menger em 1927 [Göring 2000]. O teorema caracteriza a

conectividade de um grafo através de suas conexões de arestas ou através das conexões de vértices.

A versão do teorema para conectividade de arestas é o que segue:

- Para um grafo finito não-direcionado G e um par de vértices distintos x e y, o tamanho mínimo de arestas que precisam ser removidas para desconectar x e y é igual ao número máximo de caminhos aresta-independentes emparelhados de x para y

- Extensão para subgrafos: um subgrafo maximal desconectado por não menos que k-arestas de corte é idêntico ao subgrafo maximal com um número mínimo de k caminhos aresta-independente entre qualquer par de nós x, y no subgrafo.

Já para a conectividade de vértices o teorema é apresentado na forma:

- G é um grafo finito não direcionado e x e y são dois vértices não adjacentes. Então o teorema afirma que o número mínimo de vértices que precisam ser removidos para desconectar x e y é igual ao número máximo de caminhos vértice-independentes emparelhados de x para y.

2.1.3 Design de Níveis

O design de níveis trata-se da disciplina do desenvolvimento de jogos que envolve a criação de níveis, locais, estágios ou missões [Byrne 2005]. A preocupação com design de níveis em jogos eletrônicos existe de forma explícita desde a criação dos *Multi-User Dungeons* (ou **MUD**s) [Bartle 2004].

O processo de design de níveis para cada nível individual em jogos modernos tipicamente começa com arte conceito, rascunhos, renderizações e modelos físicos. Quando esse processo é concluído, o material conceitual é transformado em documentação, modelagem de ambiente e o posicionamento de entidades específicas geralmente com um auxílio de um editor de níveis [Oxland 2004].

Os passos gerais incluem:

- Posicionando as características de larga escala do mapa, como colinas, cidades, salas, túneis, etc., para jogadores e inimigos navegarem;

- Determinar as condições ambientais e "regras base"como ciclo de dia e noite, clima, sistema de pontuação, armas permitidas, tipo de jogabilidade, limites de tempo, recursos iniciais, etc.;

- Especificar certas regiões onde certas atividades ou comportamentos ocorrem, como coleta de recursos, construção de bases, viagem pela água, etc.;
- Especificar partes não estáticas de um nível, como portas, chaves, botões, passagens secretas, etc.;
- Especificar a localização de diversas entidades, tal como unidades do jogador, inimigos, local de surgimento de monstros, escadas, moedas, nós de recurso, armas, *save points*, etc.;
- Especificar o local de entrada e saída para um ou mais jogadores;
- Adicionar detalhes estéticos tais como texturas específicas a um nível, sons, animações, luz e música;
- Introduzir locais de eventos pré-programados, onde algumas ações pelo jogador realizam mudanças específicas;
- Posicionamento de nós de *pathfinding* onde personagens não-jogáveis irão andar, as ações que os mesmos terão em resposta a gatilhos específicos e qualquer diálogo que possam ter com o jogador.

2.1.4 Algoritmos Genéticos

Algoritmos genéticos, nos ramos de computação e Investigação Operacional, trata-se de uma meta-heurística inspirada no processo de seleção natural de Charles Darwin, e pertence a categoria mais ampla de Algoritmos Evolucionários (AE) [Mitchell 1998]. Algoritmos Genéticos têm sido ao longo das últimas décadas uma história de sucesso dentre os vários algoritmos de inteligência artificial, devido em grande parte ao seu sucesso em resolver problemas de otimização com um campo de possíveis respostas extremamente amplo e informações incompletas ou imperfeitas.

Antes de detalhar o algoritmo, é importante notar que seu funcionamento é possível graças ao Teorema do Macaco Infinito, que nos diz que se dois eventos são estatisticamente independentes, então a chance de ambos acontecem é igual ao produto das probabilidades de cada um deles acontecerem independentemente. Isto é: se há uma (ou várias) respostas possíveis, um gerador aleatório de respostas seria capaz de gerar todas as possíveis respostas eventualmente [Borel 1913].

Na teoria de seleção natural de Charles Darwin, os três principais fundamentos necessários para que haja a evolução são [Darwin 2004]:

1 **Herança**: Deve haver um processo através do qual os filhos recebem propriedades dos pais.

2 **Variação**: Deve haver uma variedade de características presentes na população ou um meio pelo qual introduzir variação.

3 **Seleção**: Deve haver um mecanismo pelo qual alguns membros da população possam se tornar pais e passar sua informação genética e outros não.

Com base nesses princípios da Seleção Natural, é possível colocar o algoritmo genético na forma:

1 Inicializar de forma aleatória uma população P;

2 Determinar a adaptação de cada elemento da população;

3 Até que haja convergência ou o número de gerações máximo seja excedido, fazer:

   a Selecionar pais para a nova população

   b Fazer o *crossover* e gerar uma nova população

   c Operar a mutação na nova população

   d Calcular a adaptação da nova população

2.1.5 Elitismo

Uma característica que, ao mesmo tempo que resulta em resultados ótimos para os algoritmos genéticos, também é a causa de sua instabilidade em tempos de convergência, é o fato de que o algoritmo genético pode perder os melhores membros de sua população, e divergir de uma boa solução (mesmo que apenas uma solução local). Para um maior grau de estabilidade, também foi retirada inspiração da natureza ao formular o que é chamado de elitismo [Deb et al. 2002].

O elitismo é uma solução extremamente simples, trata-se simplesmente de copiar uma pequena proporção dos candidatos com maior adaptação, sem modificação alguma, na próxima geração. O que pode resultar em melhoras dramáticas no desempenho do algoritmo pelo fato de que boas soluções parciais não serão mais descartadas.

2.2 *Dungeons* e Grafos

Para unir as ferramentas de design de níveis aos algoritmos genéticos, é escolhido para uma analise mais criteriosa o *dungeon*. O *dungeon* trata-se de uma porção do mundo do jogo criada somente para um jogador ou grupo de jogadores [Carless 2004].

Ao longo da história dos jogos interativos, o maior dos pioneiros da criação desses espaços virtuais foi a série *The Legend of Zelda* [Stout 2012]. Para a análise, serão usado

os *dungeons* do *The Legend of Zelda* original, lançado em 1986, que já apresentava um grande grau de preocupação com o design de níveis, em grande parte pelos esforços de Shigeru Miyamoto, designer e diretor do jogo em questão. Nos níveis pode ser notado uma preocupação com os seguintes fatores:

- Fluxo do nível: A intenção do designer de níveis, Shigeru Miyamoto, queria transmitir aos jogadores um sentimento de exploração, o que foi realizado através de uma organização de salas de forma que o caminho crítico era linear (jogador não tinha necessariamente que passar pelo mesmo local mais de uma vez para completar um nível, evitando repetição), no entanto quase sempre haverão salas adjacentes ao caminho crítico para recompensar o senso de exploração do jogador pelo meio de itens entre outros.

- Intensidade crescente: A intensidade da experiência se torna mais intensa gradativamente, os monstros ficam mais difíceis e o jogador tem a chance de aprender como os inimigos funcionam em um ambiente controlado e posteriormente têm a chance de testar as habilidades aprendidas. Nos níveis analisados, há uma clara ascensão no nível de intensidade ao longo do caminho crítico, portanto fora do mesmo a intensidade apresenta variações mais arbitrárias.

- Variedade: Há uma variedade de jogabilidade, os encontros são variados, e os espaços são organizados de forma a evitar repetição. Devido às restrições de hardware da época, a variedade era limitada.

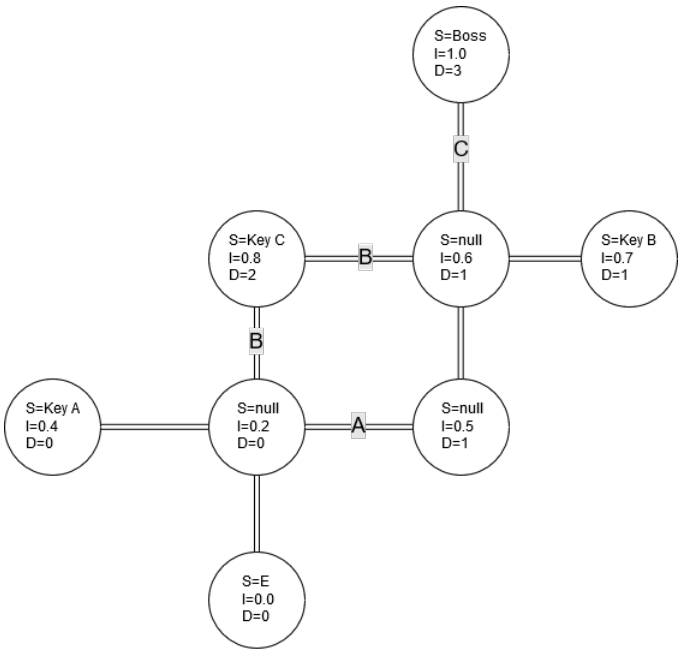
- Treinamento: Quando novas habilidades são exigidas do jogador, essas habilidades são passadas de forma controlada.

Com isso em mente, é possível começar a abstrair a composição de um *dungeon* na forma de grafos. Para a construção de grafos de *dungeon* é determinado os paralelos de que um nó representa uma sala, uma conexão a sala adjacente é representado por uma aresta, a intensidade será representada por um valor numérico variando de 0 (sala inicial) a 1 (sala onde o *boss* do *dungeon* estará presente), cada uma das salas, ou nós, irá conter um símbolo, seja ele chave, alavanca ou vazio, as conexões, ou arestas, também irão conter informação, que irão informar se a conexão é aberta, ou se tem alguma condição de que deve ser satisfeita para ser aberta.

A partir da formatação acima, a Figura 3 se torna uma clara representação de um *dungeon*, tal que é possível ver as salas como nós, e o conteúdo referente as conexões entre as salas se tornam arestas de grafo. Para isso há alguns mais atributos, cada uma das salas terá um id único, e uma posição no grafo. Além disso, temos o valor **D** que irá representar a "profundidade", isto é, quantos pré-requisitos devem ser cumpridos para se chegar nessa

sala. O símbolo **E** irá apontar onde é a entrada do *dungeon* que o jogador irá iniciar a fase, e além disso o *boss* servirá como sala final.

Figura 3 – *Dungeon* na forma de grafo



Fonte: Elaborada pelo autor.

Outro fator importante a se notar na representação de grafos do *dungeon*, é que é possível notar que salas conectadas com um diferente valor de **D** terão um pre-requisito necessário na conexão equivalente a sala com o maior valor de pre-requisito. Assim é possível começar a enxergar como seria possível implementar o algoritmo genético em um grafo de *dungeon*.

Novamente em referência a Figura 3, é possível também fazer uma análise qualitativa do design de nível envolvido na construção do grafo. É possível notar que para completar o *dungeon* o jogador necessita ir da sala de entrada até a sala contendo a *Key A*, então dessa sala até a sala contendo a *Key B*, então até a *Key C*, podendo então ir até a sala contendo o *Boss*, assim concluindo o nível. O número de formas na qual esse tipo de nível pode ser organizado cresce a medida que a complexidade aumenta. Para uma geração automática e coerente de *dungeons* segmentados e complexos, os algoritmos tradicionais de geração procedural não serão o bastante, e de fato, estudos comparativos apontam as limitações desse tipo de geração [Horn et al. 2014].

2.3 O DNA de um *Dungeon*

Para que seja possível aplicar os algoritmos genéticos em grafos, várias considerações quanto a forma de organização de grafos devem ser feitas. Primeiramente é necessário que

haja um paralelo entre os hiper-parâmetros de um algoritmo genético e os componentes de um grafo. Isso pode ser feito da forma:

- Um grupo de *dungeons* será equivalente a uma população.
- Cada *dungeon* em particular (seu conjunto de salas) será equivalente a um cromossomo.
- As salas individuais, e todo o seu conteúdo, será enxergado como sendo um gene.

Para que seja feito manipular os genes através da mutação, é necessário também abstrair o conteúdo das salas em si, portanto o conteúdo pode ser enxergado como os atributos contidos no nó. Assim é possível dar aos nós os seguintes atributos:

- id - valor inteiro contendo identificação única a cada sala (nó)
- posição - coordenadas xy da sala
- conexões - lista contendo todas as arestas da sala
- profundidade - valor inteiro contendo quantos pre-requisitos são necessários para se chegar nessa sala
- símbolo - valor que informa o que a sala irá conter, somente inimigos ou uma chave
- intensidade - valor que varia de 0 a 1 informando qual a intensidade alvo da sala, que irá por sua vez determinar o número e dificuldade do desafio apresentado pela sala

E para as arestas:

- salas - contém os ids das salas que a aresta conecta
- posição - coordenadas xy da conexão (determinado pelas coordenadas das salas conectadas)
- símbolo - contem a trava (ou ausência dela) que será determinada pelos níveis das salas conectadas caso forem de profundidade desiguais

Tendo todos os parâmetros, é possível já, em um nível teórico, aplicar o algoritmo genético ao grafo imaginado. As mutações podem ser aplicados aos parâmetros dos nós e arestas do grafo, e as avaliações de adaptação podem ser avaliadas pelos diferentes critérios estabelecidos de design de níveis [Canossa e Smith 2015]. De fato, o uso de algoritmo genético para otimização de tipos específicos de grafos já foi utilizado nas mais diversas áreas, variando de redes neurais [Stanley e Miikkulainen 2002] a sistemas de recomendação [Silva et al. 2010].

3 Modelos Generativos

No ramo de classificação estatística, o que inclui o ramo de aprendizagem de máquina, duas das principais abordagens são a abordagem discriminatória e a abordagem generativa. Elas computam classificadores através de diferentes abordagens e degrau de modelagem estatística.

3.1 Definições

Embora a terminologia seja ativamente debatida [Jordan e Mitchell 2015, Jebara 2012, Ng e Jordan 2002], para o trabalho em questão iremos usar os três grades tipos como definidos por Jebara:

1 Dado uma variável observável X e uma variável alvo Y, um modelo generativo é um modelo estatístico da distribuição de probabilidade conjunta em $X \times Y, P(X, Y)$;

2 Um modelo discriminativo é um modelo da probabilidade condicional de um alvo Y, dada uma observação x, na forma simbólica, $P(Y|X = x)$;

3 Classificadores computados sem usar um modelo probabilístico também são denominados como discriminadores de forma mais abrangente.

Em outras palavras, um modelo probabilístico é um modelo da probabilidade condicional de um observável X, dado um alvo y, o que na forma simbólica é $P(X|Y = y)$. Ou seja, um modelo generativo pode ser definido de forma simétrica a um modelo discriminativo. O termo "modelo generativo"também é usado para descrever modelos que geram instâncias de variáveis de saída de forma que a uma relação com distribuições probabilísticas entre potenciais amostras de entrada não é clara. As Redes Generativas Antagônicas (GANs) são exemplos desse tipo de modelo, pois são julgadas pela similaridade de saídas particulares em relação a entradas em potencial.

3.2 Redes Generativas Antagônicas

Primeiramente introduzidas por Ian Goodfellow em 2014 [Goodfellow et al. 2014], as GANs pertencem ao subconjunto de algoritmos que se enquadram em modelos generativos. Além disso, esses algoritmos são do campo da aprendizagem não-supervisionada, um sub-conjunto de algoritmos de aprendizagem de máquina que visam aprender a estrutura implícita de um conjunto de dados sem ter de forma explicita um valor alvo.

Esse modelo funciona a partir da simulação de um jogo entre dois jogadores. Um desses jogadores é o **gerador**, ele irá tentar criar amostras que são similares a amostras reais (como de, por exemplo, imagens) de uma distribuição de conjunto de treinamento p_{data}. O outro jogador será seu adversário (dando assim o nome ao modelo), o adversário trata-se do **discriminador**, que tenta diferenciar entre as amostras criadas pelo gerador e amostras reais do conjunto de treinamento. Então de forma geral o gerador tenta "enganar"o discriminador, fazendo-o pensar que a data gerada é real enquanto o discriminador tenta determinar de maneira precisa quais amostras são reais e quais são falsas.

Formalmente, o gerador é definido como:

$$X = G(z, \theta_G)$$

Onde X é a amostra produzida e θ_g são os parâmetros caracterizando o modelo generativo. A entrada do gerador é uma amostra de ruído z que é retirado de uma distribuição de ruído p_z. Normalmente, é utilizada a distribuição Gaussiana padrão [Weisstein 2002] ou uma distribuição uniforme para fazer a amostragem de ruído.

Já para o discriminador, tem-se:

$$D(x, \theta_D)$$

Onde θ_D são os parâmetros respectivos do modelo discriminativo e x é ou uma amostra real $x \sim p_{data}$ ou uma amostra gerada $x \sim p_G$.

O jogo então pode ser expressado como um jogo de soma-zero [Nash 1951] onde a função de *payoff* do discriminador é dada por $V(\theta_G, \theta_D)$ e o *payoff* respectivo para o gerador é o oposto, ou seja $-V(\theta_G, \theta_D)$.

A ideia central da GAN então consiste em treinar duas redes neurais diferentes para competir uma com a outra, ambas tendo diferentes funções de objetivo baseadas no valor de V. O gerador G irá tentar fazer com que a entrada gerada por ela seja reconhecida como real por D, enquanto D tentará diferenciar o verdadeiro de real, quando a rede D identificar $x \sim p_G$ como **falso**, G irá passar por um algoritmo de aprendizagem para gerar algo mais parecido com $x \sim p_{data}$, pois D foi o vencedor. Já se D identificar $x \sim p_G$ como **verdadeiro**, então G venceu a jogada e D irá passar pelo processo de aprendizagem para distinguir melhor da próxima vez. Esse processo é conhecido como treinamento antagônico, e ele ocorre até que seja estabelecido o equilíbrio de Nash.

O processo de treinamento então será, de forma passo a passo:

1 É recolhido valores de ruído de uma distribuição aleatória, então ele será usado de entrada para o gerador (G) para produzir $x \sim p_G$ com o rótulo $y = 0$ associado, que

será o par entrada-rótulo de G.

2 Uma amostra real será retirada do banco de dados (ou seja, $x \sim p_{data}$) e associada a um rótulo $y = 0$, e tanto o par real quanto o par gerado por G serão inseridos como entrada no descriminador (D) alternadamente.

3 D trata-se de uma rede neural de classificação binária, então a mesma calcula a perda para tanto $x \sim p_{data}$ quanto $x \sim p_G$, as perdas serão combinadas para ter a perda total de D.

4 G também calcula a sua perda em relação ao ruído, esse valor é a perda de G visto que cada rede tem uma função objetivo diferente.

5 As duas redes são realimentadas com suas perdas para aprender com as perdas, e assim ajustar seus parâmetros.

6 É aplicado algum algoritmo de otimização (gradiente descendente, ADAM, retro-propagação, etc.) [Hecht-Nielsen 1992, Kingma e Ba 2014, Le et al. 2011, Bottou 1991] e o processo é repetido por um determinado numero de iterações.

Pelo fato de que a função objetivo de cada uma das redes é antagônica, quando ambas estão treinando, isso ocorre de maneira a "combater"diretamente a melhorias da rede adversária. A rede G fica cada vez melhor em gerar valores próximos aos reais enquanto D fica cada vez melhor em identificar qual amostra é real e qual é falsa.

3.2.1 Modelagem Matemática

Com o modelo conceitual em mente, é possível transformar o modelo em um modelo propriamente matemático, começando pelo discriminador. Como D é um classificador binário, ao alimentá-lo com amostras de dados reais, o modelo deve produzir uma alta probabilidade para dados reais, e inversamente deve produzir baixas probabilidades quando alimentado com dados falsos (ou seja, dados gerados pela rede G).

Isso nos permite definir as variáveis e funções:

- $z \rightarrow$ vetor de ruído
- $G(z) \rightarrow$ saída do gerador $\rightarrow x_G$
- $x \rightarrow$ amostra de treinamento $\rightarrow x_R$
- $D(x) \rightarrow$ saída do discriminador para $x_R \rightarrow P(y|x_R) \rightarrow \{0, 1\}$
- $D(G(z)) \rightarrow$ saída do discriminador para $x_G \rightarrow P(y|x_G) \rightarrow \{0, 1\}$

Como é possível notar, $D(x), D(G(z))$ tem um resultado entre 0 e 1, o objetivo do modelo do discriminador é um tal que minimiza o resultado para dados falsos e maximiza para dados verdadeiros. $G(z)$ sempre terá um resultado no mesmo formato que x_R, isto é, se os dados consistem de imagens RGB 32x32, $G(z)$ irá ter um formato 32x32x3 (para os canais de cor). O objetivo do modelo de G é um tal que $D(G(z))$ seja máximo, assim temos, do lado do discriminador:

$$D_{perda_R} = \log(D(x))$$

$$D_{perda_G} = \log(1 - D(G(z)))$$

$$D_{perda} = D_{perda_R} + D_{perda_G} = \log(D(x)) + \log(1 - D(G(z)))$$

O custo total em D é:

$$\frac{1}{m}\sum_{i=1}^{m} \log(D(x^i)) + \log(1 - D(G(z^i)))$$

Já para o gerador, teremos:

$$G_{perda} = \log(1 - D(G(z)))$$

Resultando no custo total para G:

$$\frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(z^i)))$$

Com isso é possível notar que a função D é calculada duas vezes para cada vez que G é calculada, em termos práticos isso significa que para cada vez que a rede geradora é executada, a rede discriminadora será executada duas vezes, uma vez para uma entrada real, outra vez para uma entrada gerada.

As decisões do discriminador D são mais precisas uma vez que se maximiza $\mathbb{E}_{x\sim p_R(x)}[\log D(x)]$, porém quando dada uma amostra gerada $G(z), z \sim p_z(z)$, o discriminador é esperado produzir uma probabilidade, $D(G(z))$, próxima de zero, o que ocorre maximizando o valor esperado $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$. Sendo assim, para D, é possível definir a função valor V:

$$\max_D V = \mathbb{E}_{x\sim p_R(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

Já para o gerador G, o treinamento é feito com o objetivo de que D resulte em uma alta probabilidade quando analisando uma amostra gerada, portanto o objetivo é minimizar o valor esperado $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$, o que nos dá:

$$\min_G V = \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2}$$

Quando os aspectos de D e G presentes nas Equações (1) e (2) são unidos através do jogo de minimax, a função de perda a ser otimizada é a que segue:

$$\min_G \max_D \mathcal{L}(D, G) = \mathbb{E}_{x\sim p_R(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{3}$$

Portanto, através das Equações (2) e (3) é possível notar que $\mathbb{E}_{x\sim p_R(x)}[\log D(x)]$ não terá impacto algum no gradiente de G ao longo da execução do jogo.

Com uma função de perda L bem definida, é possível então determinar os valores ótimos a serem alcançados, começando por encontrar qual o valor ótimo para D.

$$\mathcal{L}(G, D) = \int_x (p_r(x)\log(D(x)) + p_G(x)\log(1 - D(x)))dx$$

Visto que o que interessa é o melhor valor de $D(x)$ para maximizar $L(G, D)$, podemos nomear:

$$\tilde{x} = D(x), A = p_R(x), B = p_G(x)$$

E então, o que está no interior da integral (a integral em si pode ser ignorada visto que x será amostrado em todos os possíveis valores) é:

$$f(\tilde{x}) = A\log\tilde{x} + B\log(1 - \tilde{x})$$

$$\frac{df(\tilde{x})}{d\tilde{x}} = A\frac{1}{\ln 10}\frac{1}{\tilde{x}} - B\frac{1}{\ln 10}\frac{1}{1-\tilde{x}} = \frac{1}{\ln 10}\left(\frac{A}{\tilde{x}} - \frac{B}{1-\tilde{x}}\right) = \frac{1}{\ln 10}\frac{A - (A+B)\tilde{x}}{\tilde{x}(1-\tilde{x})}$$

Portanto se $\frac{df(\tilde{x})}{d\tilde{x}} = 0$, o melhor valor do descriminador será obtido:

$$D^*(x) = \tilde{x}^* = \frac{A}{A+B} = \frac{p_R(x)}{p_R(x) + p_G(x)} \in [0, 1].$$

Supondo que o gerador chegue no seu valor ótimo ($p_G = p_R$), então o valor de $D^*(x)$ se torna 1/2. Assim, se tanto G e D estão nos seus valores ótimos, a função de perda se torna:

$$\begin{aligned}\mathcal{L}(G, D^*) &= \int_x (p_R(x)\log(D^*(x)) + p_G(x)\log(1 - D^*(x)))dx \\ &= \log\frac{1}{2}\int_x p_R(x)dx + \log\frac{1}{2}\int_x p_G(x)dx \\ &= -2\log 2\end{aligned}$$

Sabendo quais os valores devem ser alcançados, basta agora mostrar os gradientes de atualização de G e D para que seja possível a execução de toda a arquitetura GAN. Para o discriminador, devemos ascender o gradiente estocástico:

$$\nabla_{\theta_D}\frac{1}{m}\sum_{i=1}^{m}[\log D(x^{(i)} + \log(1 - D(G(z^{(i)})))]$$

E para o gerador, atualizamos por descender no gradiente estocástico:

$$\nabla_{\theta_G}\frac{1}{m}\sum_{i=1}^{m}[\log(1 - D(G(z^{(i)})))]$$

3.2.2 Algoritmo de Treinamento da GAN

O algoritmo originalmente desenvolvido por Ian Goodfellow para o treinamento de GANs é o que segue:

while *iterações de treino* $< n$ **do**
  while *passos* $< k$ **do**
    amostrar m exemplares de ruído $z^{(1)}, ..., z^{(m)}$ gerados através de $p_G(z)$
    amostrar m exemplares reais $x^{(1)}, ..., x^{(m)}$ dos dados reais
    atualizar D por gradiente ascendente estocástico:

$$\nabla_{\theta_D}\frac{1}{m}\sum_{i=1}^{m}[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

    passos = passos +1
  end
  amostrar m exemplares de ruído $z^{(1)}, ..., z^{(m)}$ gerados através de $p_G(z)$
  atualizar o gerador por gradiente descendente estocástico:

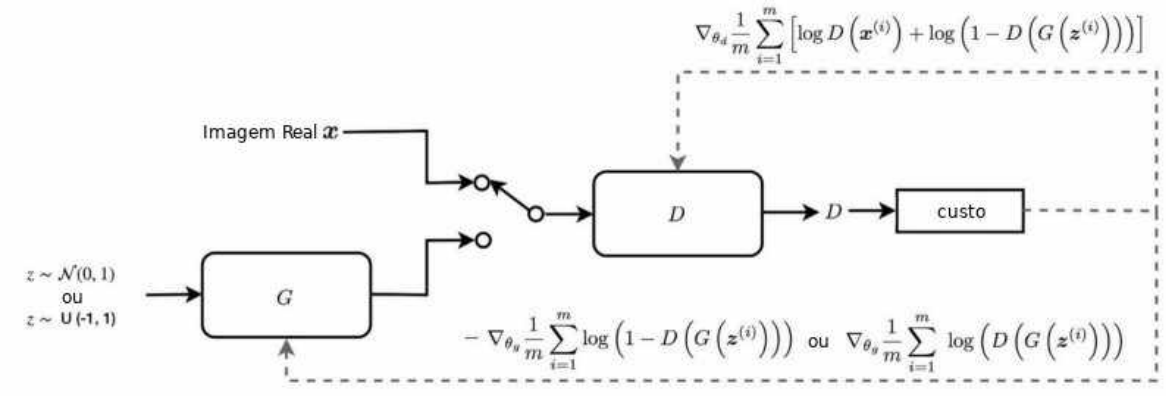
$$\nabla_{\theta_G}\frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z^{(i)})))$$

  iterações de treino = iterações de treino +1
end

O algoritmo pode ser visualizado no diagrama mostrado na Figura 4, onde D irá decidir qual imagem entre as 2 recebidas será identificada como real e a partir disso a

equação de custo é calculada e os gradientes de atualização são calculados e propagados em suas respectivas redes.

Figura 4 – Diagrama de funcionamento do algoritmo de treinamento da GAN



Fonte: Elaborada pelo autor.

3.2.3 Limitações da arquitetura GAN

Embora esse tipo de algoritmo tenha obtido um certo grau de sucesso na criação de imagens, alguns problemas teóricos, que têm consequências práticas, foram encontrados a medida que o algoritmo foi testado e desenvolvido.

Um dos maiores problemas encontrados com o treinamento de GANs baseado em gradiente descendente é o fato de que é difícil alcançar o equilíbrio de Nash [Salimans et al. 2016]. Como ambas as partes do modelo atualiza seu custo independentemente um do outro, a atualização dos gradientes não irá garantir uma convergência.

Em um jogo não-cooperativo que está buscando o equilíbrio de Nash, é simples ilustrar o porque de ser difícil de encontrar esse equilíbrio. Suponha que um jogador tome controle de x para minimizar $f_1(x) = xy$, enquanto ao mesmo tempo o outro jogador consistentemente atualiza y para minimizar $f_2(y) = -xy$.

Como $\frac{\partial f_1}{\partial x} = y$ e $\frac{\partial f_2}{\partial y} = -x$, x seria atualizado com $x - \eta y$ e y com $y + \eta x$ simultaneamente em uma única iteração, onde η é a taxa de aprendizagem. Uma vez que x e y tem sinais diferentes, cada atualização de gradiente causa uma grande oscilação, e a instabilidade se torna pior ao passar do tempo.

Outro problema encontrado nas GANs [Arjovsky e Bottou 2017, Arjovsky, Chintala e Bottou 2017], é o fato de que quando o discriminador se torna próximo de perfeito teremos $D(x) = 1, \forall x \in p_R$ e $D(x) = 0, \forall x \in p_G$, o que resulta na função L decaindo para zero, então nas iterações de aprendizagem não haverão gradientes para usar durante as atualizações.

O resultado disso é que as GANs se deparam com um dilema que torna o seu treinamento uma tarefa muito delicada:

- Se o discriminador não for adequado, o gerador não terá o *feedback* necessário e a função de perda não representa a realidade
- Se o discriminador é extremamente apto a realizar sua tarefa, o gradiente da função de perda se aproxima de zero e a aprendizagem se torna extremamente lente ou nula.

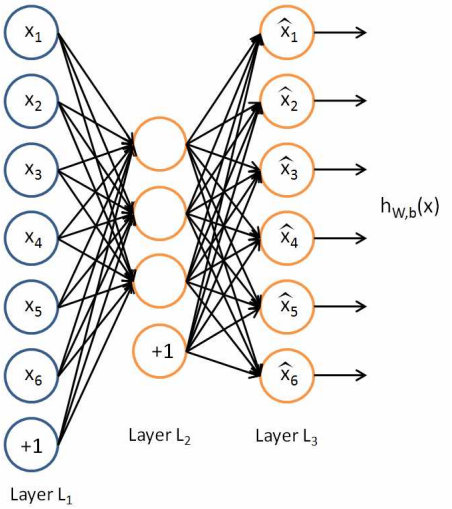
3.3 Autoencoders

Os autoencoders são um tipo de rede neural artificial que buscam aprender a representação (ou *encoding* para um grupo de dados, o que é usado tipicamente para a redução de dimensionalidade, treinando a rede para ignorar os "ruídos"de um determinado sinal [Liou, Huang e Yang 2008, Liou et al. 2014].

Além da redução de dimensionalidade, o autoencoder também aprende a reconstruir o sinal original a partir de uma representação reduzida. Essa segunda característica dos autoencoders tornou desse conceito um dos modelos generativos mais utilizados nos últimos anos [Kingma e Welling 2013].

Supondo que tenhamos amostras de treinamento $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots\}$ onde $x^{(i)} \in \mathbb{R}^n$, o objetivo do algoritmo do autoencoder será treinar, de forma não supervisionada, uma função $h_{W,b}(x)$ com um alvo $y^{(i)} = x^{(i)}$ tal que $h_{W,b}(x) \approx x$. Em outras palavras, o autoencoder irá tentar aprender uma aproximação da função identidade.

Figura 5 – Configuração geral de um autoencoder com *bias*.



Fonte: UFLDL, Stanford[2].

O autoencoder na sua forma mais simples é uma rede de *feedforward*, não distinta aos perceptrons de uma camada que participam em uma rede do tipo Perceptron Multi-Camada (MLP), isto é: uma camada de saída, uma camada entrada e uma camada oculta.

[2] Disponível em: <http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/> Acesso em maio de 2019.

Os perceptrons, por sua vez, por terem sido extensamente documentados, implementados e discutidos [Rosenblatt 1961, Rumelhart, Hinton e Williams 1985, Cybenko 1989, Gardner e Dorling 1998] estão fora do escopo do trabalho e portanto o conhecimento de seu funcionamento será presumido ao desenvolver a modelagem dos autoencoders.

Como visto na Figura 5, o propósito geral da rede será minimizar a diferença entre entrada e saída no lugar de prever um y dado um x. Como mencionado anteriormente o autoencoder consiste de duas partes, o encoder e o decoder, que podem ser definidos como transições ψ (encoder) e ϕ (decoder), de tal forma que:

$$\phi : \mathbf{X} \to \mathcal{F}$$
$$\psi : \mathcal{F} \to \mathcal{X}$$
$$\phi, \psi = \arg\min_{\phi,\psi} ||\mathbf{X} - (\psi \circ \phi)\mathbf{X}||^2$$

O treinamento dos autoencoders é, em princípio, simples, pois visa reduzir os erros de reconstrução (que também serão denominados como perda). Supondo o caso mais simples onde só há uma camada oculta, a etapa de encoder do autoencoder recebe uma entrada $x \in \Re^Q = \mathbf{X}$ e mapeia esses valores para $z \in \Re^R = \mathcal{F}$. Com uma matriz de pesos W e um *bias* b, z é calculado da forma:

$$z = \sigma(Wx + b)$$

O z resultante é normalmente referido como representação latente. Nesse caso σ trata-se da função de ativação do elemento tal como a função sigmoidal ou a função do retificador linear. Já na etapa do decoder, o autoencoder irá mapear z a uma reconstrução x' que será da mesma forma que x. Para o cálculo de x será usado um novo bias b', uma nova função de ativação σ' e uma nova matriz de pesos W, resultando em:

$$x' = \sigma'(W'z + b')$$

Após o cálculo de x' é possível calcular a perda, o que pelo método da redução dos erros quadráticos, será calculada como:

$$\mathcal{L}(x, x') = ||x - x'||^2 = ||x - \sigma'(W'(\sigma(Wx + b)) + b')||^2$$

3.3.1 Autoencoder Variacional

Os Autoencoders Variacionais (VAEs), são uma implementação dos modelos mais gerais de variáveis latentes contínuas [Doersch 2016] que são capazes de serem usados em uma grande variedade de aplicações, como geração de imagens, vídeos e formas.

Em geral, o objetivo de um modelo de variável latente contínua é aprender um espaço latente $Z = \Re^Q$ dado um conjunto de amostras $\{y_m\} \subseteq Y = \Re^R$ onde $Q < R$, isto é, uma redução de dimensionalidade.

Esse modelo, em particular, consiste em dois componentes, o modelo generativo $p(y|z)$ dado um valor prévio fixo $p(z)$, e o modelo de reconhecimento (ou inferência) $q(z|y)$. No caso das VAEs, tanto os modelos de geração quanto de inferência são implementados usando redes neurais artificiais. Assim, tanto o modelo de reconhecimento quanto a probabilidade marginal podem ser somente aproximadas, o que é garantido pela equação abaixo:

$$p(y) = \int p(y, z)dz = \int p(y|z)p(z)dz \tag{4}$$

Seguindo a Equação (4), é possível fazer o uso inferência variacional [Blei, Kucukelbir e McAuliffe 2017] para maximizar o limite inferior da probabilidade. Em geral, a inferência variacional é tida como o problema de encontrar o modelo de distribuição $q(z)$ para aproximar o verdadeiro posterior $p(z|y)$ onde o a divergência de Kullback-Leibler **KL** é uma medida de distância definida nas distribuições probabilísticas. A divergência de Kullback-Leiber pode então ser reescrita para obter um limite inferior da probabilidade marginal $p(y)$.

Essa divergência entre duas distribuições probabilísticas $q(z)$ e $p(z|y)$ é definida formalmente como:

$$\mathbf{KL}(q(z)|p(z|y)) = \mathbb{E}_{q(z)}\left[\ln \frac{q(z)}{p(z|y)}\right] \tag{5}$$

onde $\mathbb{E}_{q(z)}$ denota a expectativa em respeito a distribuição $q(z)$.

Através da Equação (5), é possível formular a distribuição $q(z)$ na forma de um problema de otimização da forma:

$$q(z) = \arg\min{}_q \mathbf{KL}(q(z)|p(z|y)) \tag{6}$$

Através da equação (5), é revelado que o problema de otimização da equação (6) necessita computar a probabilidade marginal:

$$\mathbf{KL}(q(z)|p(z|y) = \mathbf{E}_{q(z)}\left[\ln \frac{q(z)}{p(z|y)}\right]$$

$$= \mathbf{E}_{q(z)}[\ln q(z)] - \mathbf{E}_{q(z)}[\ln p(z|y)]$$

$$= \mathbf{E}_{q(z)}[\ln q(z)] - \mathbf{E}_{q(z)}[\ln p(z, y)] + \ln p(y)$$

Reorganizar ambos os lados da equação resulta no limite inferior variacional:

$$\begin{aligned}\ln p(y) &= \mathbf{KL}(q(z)|p(z|y)) - \mathbf{E}_{q(z)}[\ln q(z)] + \mathbf{E}_{q(z)}[\ln p(z,y)] \\ &\geq -\mathbf{E}_{q(z)}[\ln q(z)] + \mathbf{E}_{q(z)}[\ln p(z,y)] \\ &= -\mathbf{E}_{q(z)}[\ln q(z)] + \mathbf{E}_{q(z)}[\ln p(z)] + \mathbf{E}_{q(z)}[\ln p(y|z)] \\ &= -\mathbf{KL}(q(z)|p(z)) + \mathbf{E}_{q(z)}[\ln p(y|z)]\end{aligned}$$

O problema original de maximizar a probabilidade marginal $p(y)$ na Equação (4) é então aproximado através da maximização do limite inferior variacional, sendo o mesmo formalmente definido através da equação:

$$\mathbf{E}_{q(z)}\left[\ln \frac{q(z)}{p(z|y)}\right] = -\mathbf{KL}(q(z)|p(z)) + \mathbf{E}_{q(z)}[\ln p(y|z)] \tag{7}$$

Nessa formulação geral do limite inferior variacional, a distribuição do modelo $q(z)$ poderá ser arbitrária. No entanto, em um contexto de modelos variacionais latentes, é recomendado que a distribuição do modelo dependa de y de forma explícita, i.e. $q(z|y)$, visto que é desejada a reconstrução de qualquer y a partir de um código correspondente z.

O limite inferior variacional da Equação (7) toma a forma de um autoencoder onde $q(z|y)$ representa o encoder e $p(y|z)$ representa o decoder. Assim, o VAE pode ser treinado através da maximização do lado esquerdo da Equação (7) escolhendo parametrizações adequadas para as distribuições $p(z)$ e $q(z|y)$.

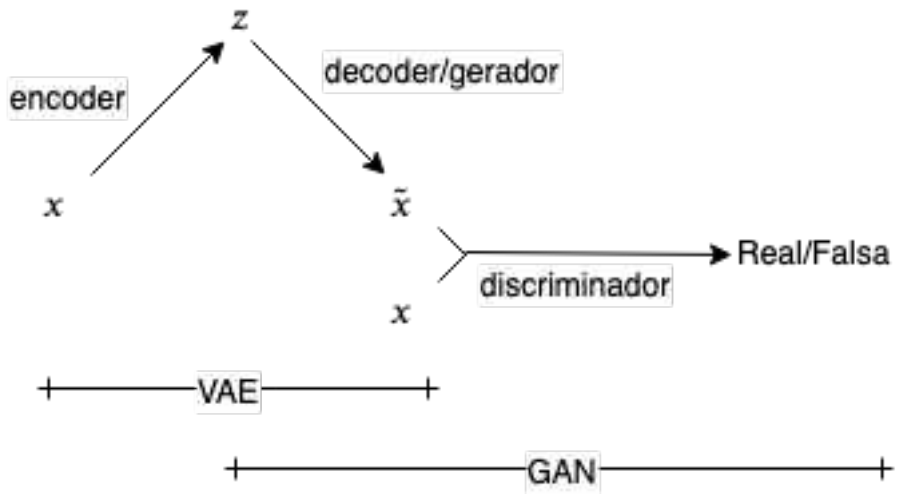
3.4 VAEGANs

Um dos problemas encontrados nas VAEs é de que a escolha de métrica de medida de erros é crucial para o treinamento da arquitetura. Para tarefas como a geração de imagens, métricas de medida de erro pixel-a-pixel, embora simples, não são ideais pois não modelam as propriedades da percepção visual humana. Esse problema resulta em traduções de imagens que podem apontar grandes erros a nível de pixel que não seriam significantes para a visão humana, e ao mesmo tempo favorecer imagens borradas que seriam claramente percebidas por um observador humano. Isso ocorre pelo fato de que as VAEs favorecem distribuições normais (também chamadas de distribuições Gaussianas), e as perdas pixel-a-pixel não capturam as diferenças em percepção e correlação espacial entre imagens [Leachtenauer et al. 1997, Zhao, Song e Ermon 2017].

Uma solução proposta é usar do discriminador de uma GAN para tornar a forma de validação de saída das VAEs mais similar a operações cognitivas de alto nível feita por humanos ao inspecionar uma imagem. Isso é feito através da substituição do gerador G de uma GAN pela rede do tipo VAE [Larsen et al. 2015, Mescheder, Nowozin e Geiger 2017], o que pode ser visto na Figura 6.

Uma das propriedades mais interessantes das GANs é que a sua rede discriminadora deve aprender uma métrica de similaridade extremamente complexa para imagens de forma implícita, de forma a distinguir as imagens reais de imagens geradas. A proposta das VAEGANs consiste em explorar esse fato de forma a transferir as propriedades das imagens aprendidas pelo discriminador em um valor de erro de reconstrução mais abstrato para o VAE. O resultado final é um método que combina as vantagens de uma GAN como um modelo generativo de alta qualidade com as da VAE como um método que produz um codificador de dados a um espaço latente z.

Figura 6 – Arquitetura de uma VAEGAN.



Fonte: Elaborada pelo autor.

De forma específica, como erros de reconstrução restritos a elementos individuais não são adequados para imagens e outros sinais com invariância, a VAEGAN irá substituir a sua original equação de erro por um erro de reconstrução resultante do discriminador da GAN. Para que isso seja alcançado, sendo $\mathbf{Dis}_l(x)$ a representação da camada oculta l do discriminador (que para não ser confundido com o decoder, passa a ser representado por **Dis**), é introduzido um modelo de observação Gaussiana para $\mathbf{Dis}_l(x)$ de média $\mathbf{Dis}_l(\tilde{x})$ e covariância identidade:

$$p(\mathbf{Dis}_l(x)|z) = \mathcal{N}(\mathbf{Dis}_l(x)|\mathbf{Dis}_l(\tilde{x}), \mathbf{I}), \tag{8}$$

onde $\tilde{x} \sim \mathbf{Dec}(\mathbf{z})$ é a amostra do decoder de x. Torna-se então possível a substituição da equação de erro da VAE por:

$$\mathcal{L}_{\mathbf{Dis}_l} = -\mathbb{E}_{q(z|x)}\big[\log p(\mathbf{Dis}_l(x)|z)\big] \tag{9}$$

Torna-se possível então definir o modelo de treinamento a partir da combinação das perdas tal que:

$$\mathcal{L} = \mathcal{L}_{previo} + \mathcal{L}_{\mathbf{Dis}_l} + \mathcal{L}_{GAN} \tag{10}$$

A partir da combinação de perdas, os parâmetros da rede VAE serão otimizadas. $\mathcal{L}_{GAN}$ é denominado como **erro estilístico**, já o erro de reconstrução é interpretado na literatura como sendo o **erro de conteúdo** [Gatys, Ecker e Bethge 2015]), e como tanto **Dec** quanto o gerador G mapeiam de z para x. seus parâmetros serão compartilhados. Em outras palavras, G torna-se o decoder.

Usando da Equação (10), tanto a VAE quanto a GAN são treinadas de forma simultânea, isso é possível porque os parâmetros da rede não são atualizados com a perda combinada. Em particular, **Dis** deve tentar não minimizar $\mathcal{L}_{\mathbf{Dis_l}}$ pois isso resultaria em uma perda que colapsa em 0. O sinal de erro de $\mathcal{L}_{GAN}$ também não é propagado para o encoder (**Enc**).

A medida que **Dec** recebe um sinal de erro de tanto $\mathcal{L}_{\mathbf{Dis_l}}$ quanto $\mathcal{L}_{GAN}$, um parâmetro γ é usado para pesar a habilidade de reconstrução em relação a habilidade de enganar o discriminador. Isso pode também ser interpretado como uma forma de pesagem de **estilo** e **conteúdo**. Ao aplicar γ na Equação (10), a pesagem será feita somente quando atualizando os parâmetros de **Dec**:

$$\theta_{\mathbf{Dec}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathbf{Dec}}}\left(\gamma\mathcal{L}_{\mathbf{Dis_l}} - \mathcal{L}_{GAN}\right) \tag{11}$$

Essa pesagem também contribui a corrigir o problema original causado pelo fato de que distâncias de medidas dependentes de elementos individuais são notoriamente ineficientes para distribuições complexas de dados tais como imagens [Wang e Bovik 2009, Snell et al. 2017].

Ao amostrar $q(p|x)$ (i.e. o encoder **Enc**) de forma conjunta à $p(x)$ do objetivo da GAN, tem-se:

$$\mathcal{L}_{GAN} = \log\left[\mathbf{Dis}(x)\right] + \log\left[1 - \mathbf{Dis}(\mathbf{Dec}(z))\right] + \log\left[1 - \mathbf{Dis}(\mathbf{Dec}(\mathbf{Enc}(x)))\right] \tag{12}$$

A regularização do espaço latente $\mathcal{L}_{previo}$ deve tornar o conjunto de amostras de tanto $p(z)$ ou $q(z|x)$ parecidas. Entretanto, para cada dado exemplo x, é mais provável que a amostra negativa $\mathbf{Dec}(\mathbf{Enc}(x))$ seja similar a x do que $\mathbf{Dec}(\mathbf{z})$.

3.4.1 Algoritmo de Treinamento da VAEGAN

Para que haja o treinamento da arquitetura VAEGAN, conforme as equações desenvolvidas previamente, é realizado o algoritmo:

$\theta_{\mathbf{Enc}}, \theta_{\mathbf{Dec}}, \theta_{\mathbf{Dis}} \leftarrow$ inicializa os parâmetros da rede
while *iterações* $< n$ **do**
  $\mathbf{X} \leftarrow$ amostra aleatória do banco de dados
  $\mathbf{Z} \leftarrow \mathbf{Enc}(\mathbf{X})$
  $\mathcal{L}_{previo} \leftarrow D_{KL}(q(\mathbf{Z}|\mathbf{X})||p(\mathbf{Z}))$
  $\tilde{\mathbf{X}} \leftarrow \mathbf{Dec}(\mathbf{Z})$
  $\mathcal{L}_{\mathbf{Dis}_l} \leftarrow -\mathbb{E}_{q(\mathbf{Z}|\mathbf{X})}\left[p(\mathbf{Dis}_l(\mathbf{X})|\mathbf{Z})\right]$
  $\mathbf{Z}_p \leftarrow$ amostras de um prévio $\mathcal{N}(0, I)$
  $\mathbf{X}_p \leftarrow \mathbf{Dec}(\mathbf{Z}_p)$
  $\mathcal{L}_{GAN} \leftarrow \log\left[\mathbf{Dis}(x)\right] + \log\left[1 - \mathbf{Dis}(\mathbf{Dec}(z))\right] + \log\left[1 - \mathbf{Dis}(\mathbf{Dec}(\mathbf{Enc}(x)))\right]$
  //atualização dos parâmetros de acordo com os gradientes
  $\theta_{\mathbf{Enc}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathbf{Enc}}}\left(\mathcal{L}_{previo} - \mathcal{L}_{\mathbf{Dis_l}}\right)$
  $\theta_{\mathbf{Dec}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathbf{Dec}}}\left(\gamma\mathcal{L}_{\mathbf{Dis_l}} - \mathcal{L}_{GAN}\right)$
  $\theta_{\mathbf{Dis}} \overset{+}{\leftarrow} -\nabla_{\theta_{\mathbf{Dec}}}\mathcal{L}_{GAN}$
end

Através da implementação do algoritmo descrito acima, é possível, através do uso de pares de imagem idênticas, treinar o modelo para que o mesmo aprenda a gerar distribuições próximas às do branco de dados. Caso as imagens sejam diferentes, o algoritmo será treinado em um processo denominado tradução de imagens [Huang et al. 2018].

4 Desenvolvimento

O desenvolvimento foi realizado em, principalmente, duas etapas, cada uma delas contendo diversas sub-etapas. Dentre as etapas a plataforma alvo foi comum, sendo esta a plataforma de desenvolvimento de jogos Unity. O motor de jogos Unity foi escolhido pois não somente o mesmo possui um grande grau de flexibilidade para o desenvolvimento de jogos e animações em 2D [Hocking 2015, Calabrese 2014], mas também vêm se tornando uma plataforma cada vez mais apta para o desenvolvimento de aplicações de aprendizagem de máquina [Juliani et al. 2018].

4.1 Editor de Dungeon Genético

A primeira parte a ser abordada pelo trabalho será o desenvolvimento da ferramenta de edição de níveis de forma automatizada usando de algoritmos genéticos. A aplicação específica a ser desenvolvida irá ser especificadamente para permitir que o designer de níveis crie dungeons de forma automatizada, porém que diferente dos métodos de criação procedural convencionais, atenda à parâmetros qualitativos de design de jogos.

4.1.1 Levantamento de Requisitos

Para a construção de uma ferramenta de edição de níveis que será útil a um designer, mas que ao mesmo tempo permita a criação de níveis de forma rápida e semi ou totalmente automatizada, é necessário que sejam atendidos primeiramente requisitos básicos de um editor de níveis, para depois tal editor seja equipado com a capacidade de fazer a geração de forma autômata.

Para isso a ferramenta de edição deverá ser capaz de:

- Escolher o tipo de restrição que será aplicada ao dungeon

    1 Restrição Genérica - Permitirá o algoritmo de Algoritmo Genético ter como única restrição o número total de salas e de níveis.

    2 Restrição Espacial - Tipo de restrição que irá usar uma disposição espacial de salas como restrição adicional à ser usada pelo Algoritmo Genético, limitando o algoritmo em sua organização espacial.

- Caso seja escolhida a Restrição Espacial, o software deve ter uma ferramenta básica de edição de *layout* de mapa.

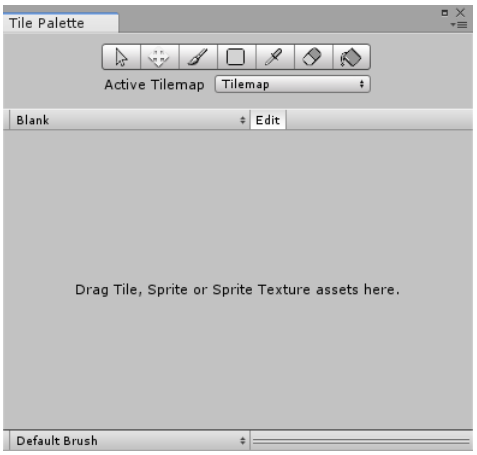
- O editor de mapa deverá contar contar com uma implementação genérica de algoritmos genéticos, capaz de receber diferentes tipos de classes como populações, cromossomos e genes, diferentes funções de mutação e função de avaliação de *fitness*. Desta forma o designer seria capaz de escolher os próprios requisitos qualitativos a serem buscados pelo algoritmo.

4.1.2 Editor de Mapa

Para a construção de um editor de mapa, visando manter a ênfase na implementação para design de níveis do algoritmo genético, as ferramentas do pacote 2D-Extras[3] foram escolhidas para ser adaptadas e modificadas de forma a torná-las um editor de níveis robusto. O pacote em questão é distribuído como complemento em fase experimental para desenvolvedores em 2D do motor de jogos Unity.

Para implementar o criador de mapas, é feita uma abstração [Szyperski, Gruntz e Murer 2002] de dois componentes já presentes nas extensões de desenvolvimento 2D do motor, para que a partir de uma reimplementação das classes já existentes seja possível ao designer criar mapas de *tiles* (que serão interpretadas como salas) de forma simples e eficiente.

Figura 7 – Editor de Tile Palette.

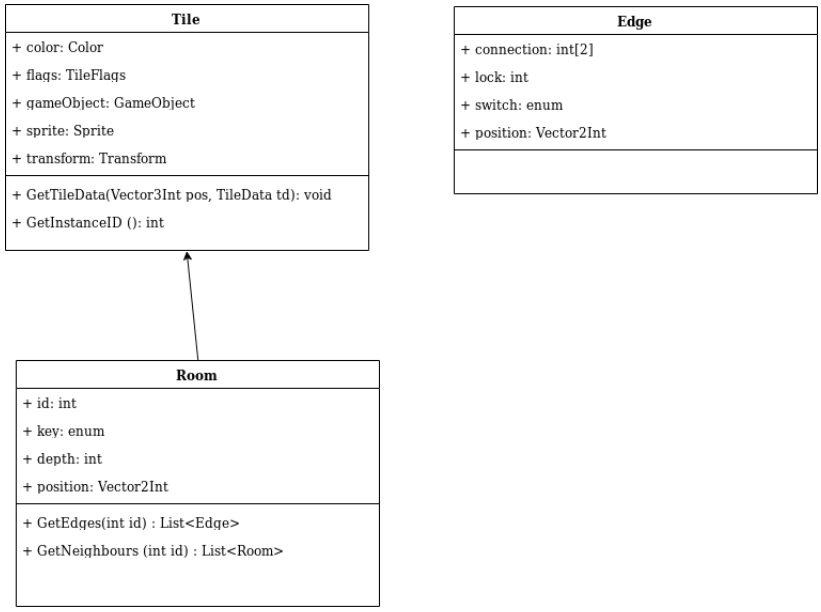


Fonte: Documentação do motor de jogos Unity.

As classes a serem abstraídas será a Tile Palette e a Tile, com a Tile Palette é possível "pintar"*tiles* na tela. Através da abstração realizada, é importante ressaltar que somente algumas das propriedades das classes serão alteradas, o que pode ser visto através do diagrama de classes apresentado na Figura 8 que segue, onde a classe criada Room irá herdar todos os métodos e parâmetros da classe Tile, porém além deles terá mais alguns métodos e parâmetros que serão fundamentais na criação de um Tilemap que possa ser usado pelo algoritmo genético.

[3] 2D-Extras, Unity Technologies – Disponível em: <https://github.com/Unity-Technologies/2d-extras> Acesso em maio de 2019

Figura 8 – Diagrama de classes.

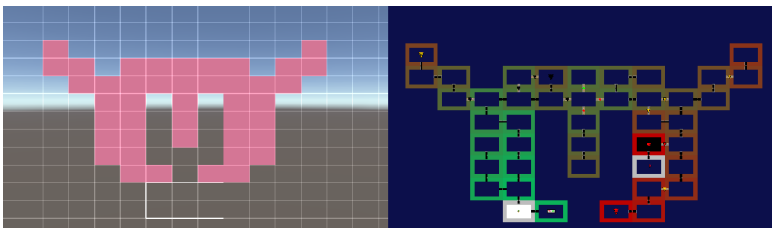


Fonte: Elaborada pelo autor.

A classe Edge contém um vetor de inteiro denominado *connection* que irá guardar dois valores, sendo estes valores os ids das salas que o mesmo conecta. Uma sala irá conter um método capaz de pegar todas os elementos da classe Edge que a conectam a alguma outra sala, assim sabendo quais salas estão ligadas a ela. A definição dessas conexões será tratada de forma algorítmica durante o processo de transformação do TileMap em grafo de *dungeon*, e depois modificado pelo algoritmo genético.

A partir do posicionamento, escolhido a critério do designer, das salas no TileMap será executado um novo algoritmo que irá escolher de forma aleatória uma sala de entrada, de saída, a posição das chaves e estado das conexões. Após a montagem do algoritmo, é possível que seja feita a criação de dungeons de forma semi-automatizada, porém sem nenhum critério de avaliação qualitativo.

Figura 9 – Editor de dungeon, com seu dungeon resultante à direita.



Fonte: Elaborada pelo autor.

Na Figura 9 é possível ver todas as representações necessárias das características do dungeon. A sua cor representa a sua intensidade (valor que vai de verde ao vermelho conforme a intensidade aumenta), suas conexões são mostradas pelas portas que existem entre elas, e a sala objetivo e de entrada são representadas, respectivamente, por um ícone

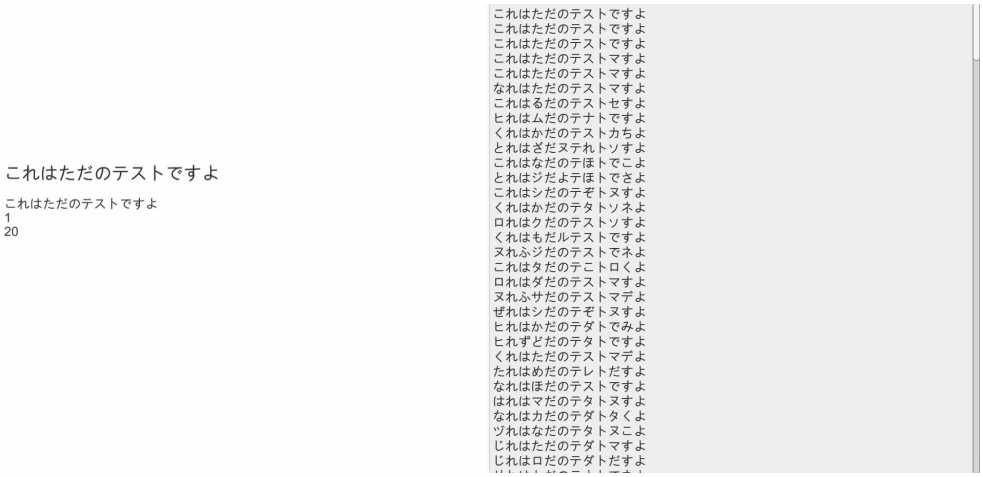
de uma bandeira e pelo ícone do personagem na sala branca. Além disso, temos as chaves em portas trancadas e em algumas salas que representam a chave necessária para abrir determinada porta ou a chave que está contida em uma sala.

4.1.3 Implementação Genérica de Algoritmo Genético

Usando dos conceitos de classes abstratas e polimorfismo [Larman 2002] e a linguagem de programação C# [Schildt 2010], é desenvolvido o algoritmo descrito no capítulo 2, já contando com a implementação do elitismo. Para manter a função de avaliação, e o algoritmo de forma geral, genérico e capaz de receber diferentes objetos e funções de avaliação, foram escolhidas para uso extensivo a biblioteca LINQ [Meijer, Beckman e Bierman 2006] e do método Func, que permite o desenvolvimento de código delegando funções que serão implementadas posteriormente.

Para o algoritmo genético, a função delegada será a de avaliação de aptidão de uma população. Ao testar a implementação foi escolhido o exemplo simples onde a partir de uma string (formada por um vetor de caracteres) foi implementado o algoritmo com elitismo para avaliar a semelhança de frases com uma frase escolhida a partir de um conjunto possível de letras. Nesse caso a função delegada de avaliação será medida por igualdade de caracteres, e a mutação ocorrerá na mudança de caracteres.

Figura 10 – Algoritmo Genético ao fim de sua execução.



Fonte: Elaborada pelo autor.

O algoritmo é testado nesta forma (vista na Figura 10) por ser uma maneira simples e visual de testar todos os componentes necessários da implementação de um algoritmo já extensamente estudado e discutido na literatura [Kramer 2017]. A população de frases irá buscar os componentes mais parecidos com o da esquerda, terão sua semelhança avaliada como aptidão, os 5 mais aptos serão clonados na próxima geração, e eles irão aparecer na forma de lista com aptidão decrescente.

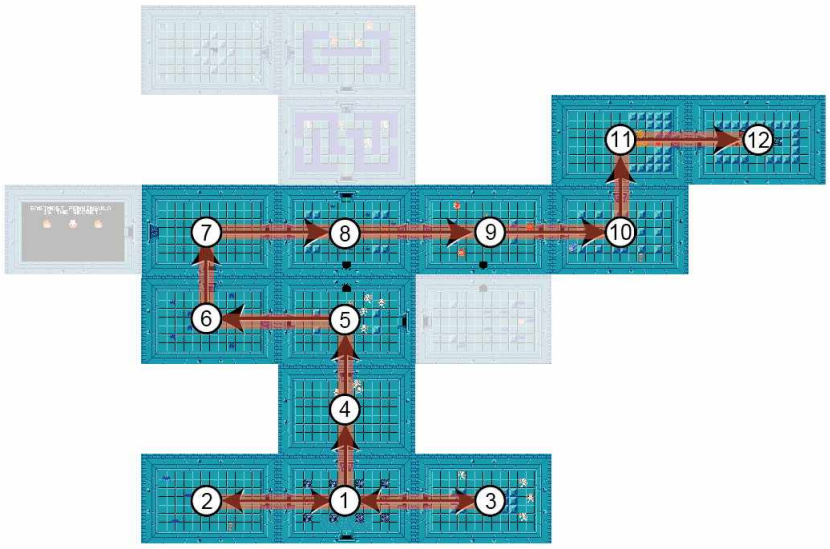
Na implementação realizada, é possível notar uma rápida convergência com apenas 20 gerações de um grupo de possíveis strings extremamente extenso devido ao número de possíveis caracteres nos alfabetos em questão [Miller 1967]. Embora o problema exemplo tenha um grau de complexidade relativamente elevado, o estudo probabilístico da organização de grafos tem um grau de complexidade muito mais elevado [Banerji, Mansour e Severini 2013] devido a natureza com a qual as conexões entre grafos e suas características alteram o espaço de soluções de formas que são não-triviais de serem analisadas de forma algébrica.

4.1.3.1 Função de Avaliação de Aptidão de um *Dungeon*

Para a implementação bem sucedida do algoritmo no contexto específico, o componente crucial a ser estudado e desenvolvido será a função de avaliação a ser usada pelo algoritmo genético. Como discutido no capítulo 2, o critério de avaliação qualitativo é arbitrário portanto depende apenas das vontades do designer. Para o trabalho sendo desenvolvido no entanto, o critério que será escolhido para avaliação qualitativa da população de *dungeons* gerados será sua **não-linearidade**.

A não-linearidade é definida pela característica de um nível no qual, para a conclusão do mesmo, é necessário o retrocesso em áreas nas quais o jogador já esteve (conhecido pelo termo *backtracking*). Para o algoritmo genético desenvolvido a busca é por **minimizar** a não-linearidade.

Figura 11 – Caminho Crítico em *dungeon* de *The Legend of Zelda*.



Fonte: STOUT, 2012[4].

[4] Disponível em: <https://www.gamasutra.com/view/feature/6582/learning_from_the_masters_level_.php> Acesso em maio de 2019.

Para a Figura 11, isso é demonstrável através da enumeração em cada uma nas salas, representando qual a ordem na qual as salas devem ser visitadas para que o nível seja concluído da forma mais eficiente possível. A partir disso é possível analisar a não-linearidade através de uma análise do caminho percorrido, onde notadamente a sala de demarcação 1 deve ser visitada em duas ocasiões para que o nível seja concluído.

4.1.3.2 *Pathfinding* e Não-Linearidade

Um dos componentes necessários para encontrar os caminhos críticos a serem avaliados é um algoritmo de *pathfinding* [Botea, Müller e Schaeffer 2004], e como um algoritmo genético com vários *dungeons* sendo testados a cada iteração requer um grande esforço computacional, a performance do algoritmo é fundamental. Para o caso específico da aplicação desenvolvida, o algoritmo A* não é ótimo, pois o grafo de *dungeon* é um grafo bidimensional onde todos os custos são idênticos, sendo este o pior cenário possível para o algoritmo em questão [Harabor e Grastien 2011].

Após uma busca na literatura, é implementada uma versão adaptada do algoritmo *Jump Point Search* (JPS) [Harabor e Grastien 2012, Harabor e Grastien 2014], proporcionando uma melhora em ordem de magnitude sobre a implementação original do algoritmo A* para o tipo de grafo em questão. A implementação irá ser executada em partes, de forma a chegar do início ao fim de cada um dos *dungeons* de uma população. A execução é dada da forma:

```
Nó atual ← Sala de entrada
while nó atual ≠ Sala de saída do
    Caminho ← JPS(nó atual, Sala Objetivo i)
    i++
end
```

Ao final da execução o caminho contém uma lista de nós que foram visitados, tornando trivial a tarefa de avaliar quais salas de um *dungeon* foram visitadas mais que uma vez ao longo do caminho crítico para se completar o nível, pois basta avaliar quantas vezes cada id encontra-se presente na lista.

Para cada repetição encontrada a variável não-linearidade é incrementada, e o algoritmo genético passa a buscar aquele com o menor valor desta variável. Completando assim a etapa de desenvolvimento do editor de níveis e também sua integração a um algoritmo genético capaz de avaliar as populações geradas de forma semi ou completamente automatizadas de acordo com parâmetros qualitativos definidos por um designer de níveis.

4.2 Geração de *Sprites*

Esta seção é dedicada a detalhar o desenvolvimento dos modelos generativos que serão implementados, assim como da criação dos bancos de dados a serem gerados. Como o objetivo do trabalho é gerar *sprites* em acordo com um estilo artístico específico, serão usadas duas bases de dados com estilos diferentes e internamente consistentes.

4.2.1 Levantamento de Requisitos

Os modelos generativos a serem desenvolvidos, de forma a se tornarem uma ferramenta útil para designers, artistas e desenvolvedores, deve contar com uma série de características. Ambos os modelos generativos, GANs e VAEGANs, irão partilhar de um número de características, no entanto suas especificações e distinções devem ser esclarecidas.

Para os modelos generativos deve-se ter:

- Para a GAN:
  1 Um banco de dados em um formato padrão deve ser organizado de forma que a rede possa usar para fazer amostragens.
  2 A rede deve ser capaz de, a partir de uma base de dados com consistência estilística interna, gerar novas imagens consistentes com o mesmo estilo.
- Para a VAEGAN:
  1 Um outro banco de dados, com pares de imagens em um formato padrão deve ser organizado para a rede fazer amostragens de forma a treinar a tradução de imagens
  2 A rede deve ser capaz de, após treinada na tradução de imagens, a partir de um determinado espaço amostral entrado pelo usuário – na aplicação específica um artista – e traduzir a imagem para o estilo desejado.

4.2.2 GAN

O desenvolvimento da GAN foi dividido em duas etapas, o desenvolvimento da arquitetura em si e a criação do banco de dados. Para testar um caso de fronteira da GAN, o estilo artístico escolhido para ser emulado foi *pixel art*. Para esse estilo artístico, particularmente comum à quarta geração de consoles [Kent 2010], um pequeno desvio de precisão torna-se claramente visível e, ao mesmo tempo, as dimensões das imagens serem reduzidas são favoráveis ao treinamento da arquitetura.

Figura 12 – *Sprites* de *Final Fantasy* a serem emulados.

Fonte: Imagem montada pelo autor[5].

Como é notável na Figura 12, as imagens dos personagens apresentados, embora possuam pixels visíveis, têm também um estilo artístico característico e consistente. O caso escolhido para formação do banco de dados e treinamento é interessante pois requer que o modelo tenha uma excelente abstração de estilo artístico e também uma ótima precisão.

O banco de dados foi gerado através de técnicas de *web scraping* [Mitchell 2018] para coletar *sprites* dos jogos Final Fantasy IV, Final Fantasy V e Final Fantasy VI, totalizando 1753 imagens no formato ARGB de dimensões 20x28px.

Para a GAN simples, basta que todas as imagens estejam em um só diretório de forma que uma imagem possa ser amostrada, pois a rede G irá produzir $G(z)$ através de um vetor de ruído retirado de um espaço com uma distribuição gaussiana de valores. Para a arquitetura da rede, serão feitas algumas modificações em relação a arquitetura do tipo DCGAN [Radford, Metz e Chintala 2015] para melhor encaixar a solução ao problema proposto.

Tabela 1 – Arquitetura da rede D.

Camada	**Operação**	**Dimensão de Saída**	**Ativação**
Entrada	Extração de valores RGB	20x28x3x64	Linear
Convolucional Transposta 1	TransConv2D	32x32x64x64	LReLU
Convolucional 1	Conv2D	16x16x128x64	LReLU
Convolucional 2	Conv2D	8x8x256x64	LReLU
Convolucional 3	Conv2D	4x4x512x64	LReLU
Camada Totalmente Conectada	FProp	1	Sigmoid

Após a montagem do banco de dados, a rede poderá ser desenvolvida e implementada. Quanto a arquitetura, a mesma será desenvolvida tendo como parâmetros de entrada o formato uniforme do banco de dados, os campos de transparência da imagem (valor de canal alpha igual a 0) serão considerados como pontos pretos, onde $RGB = (0, 0, 0)$ de forma a minimizar a quantidade de valores a serem trabalhados pela rede.

[5] Squaresoft, Final Fantasy V. Super Famicom, 1992.

Com a arquitetura definida de forma a atender os parâmetros específicos de entrada do problema, conforme visto na Tabela 1, serão amostradas 64 imagens por vez para que seja possível efetuar um processo denominado *batch normalization* [Ioffe e Szegedy 2015] que visa possibilitar uma maior estabilidade durante o processo de treinamento. Já para a rede G será necessário fazer 64 amostragens de um vetor de ruído z, de forma que teremos um $G(z)$ para cada x real. A arquitetura de G pode ser vista na Tabela 2, e além de diferenças na arquitetura entre a implementação desenvolvida e a DCGAN original, também foram usadas funções de ativação diferentes e a cada vez que D é executada, G será executada duas vezes, visando evitar que o gradiente de aprendizagem desapareça por D estar próxima de perfeita.

Tabela 2 – Arquitetura da rede G.

Camada	**Operação**	**Dimensão de Saída**	**Ativação**
Entrada	Amostras de ruído	100x64	Linear
Convolucional Transposta 1	TransConv2D	4x4x1024x64	LReLU
Convolucional Transposta 2	TransConv2D	8x8x512x64	LReLU
Convolucional Transposta 3	TransConv2D	16x16x256x64	LReLU
Convolucional Transposta 4	TransConv2D	32x32x128x64	LReLU
Camada de Saída	FProp	20x28x3x64	Tanh

A partir de uma arquitetura e modo de execução estabelecidos, é possível então implementar a rede proposta em C#, visando uma implementação final em tempo real. Para maior flexibilidade e facilidade de alterar a rede à medida do necessário, a rede é feita de forma modular para que camadas, funções de ativação e otimização sejam separadas e possam interagir de diferentes formas dependendo das definições e preferências do usuário, tornando assim trivial a alteração de uma função de ativação ou dimensão de uma camada em específico. A implementação das camadas e funções de ativação foi feita de forma similar a uma implementação em C/C++ denominada *tiny-dnn*[6], que não possui dependências, e camadas como *encoder* e *decoder* foram adicionadas, além de otimização por *batch normalization* na versão em C# desenvolvida.

Após a criação dos bancos de dados e implementação em código da GAN simples foram realizadas, inicialmente, 4000 iterações de treinamento período de pouco menos de um mês com os pesos sendo armazenados periodicamente. Os dados coletados e experimentos realizados com diferentes funções de ativação serão detalhados no Capítulo 5.

[6] Disponível em: <https://github.com/tiny-dnn/tiny-dnn> Acesso em junho de 2019.

4.2.3 VAEGAN

As VAEGANs, embora tratam-se apenas de uma GAN na qual a rede G têm uma configuração*encoder-decoder*, possuem requisitos diferentes a serem atendidos na formação do banco de dados que será utilizada. Para a implementação da VAEGAN e seu treinamento em tradução de imagens, o banco de dados deve possuir uma imagem e sua imagem traduzida correspondente.

Figura 13 – Par de imagens de treinamento.

Fonte: Elaborada pelo autor.

Como a intenção da implementação é ajudar artistas a gerarem arte em 2D, que vai desde animações e *storyboarding* até arte conceitual, o banco de dados contém pares de imagem de tal forma que um lado contém o *outline* da imagem enquanto o outro contém a imagem original. De forma que após o treinamento a VAEGAN será capaz de receber um rascunho de um desenho e transformá-lo em um desenho completo.

Para a geração do banco de dados de outlines, é necessário primeiro um banco de dados das imagens finais, e, assim como no caso anterior, o estilo artístico deverá ser consistente para que o erro estilístico visto no capítulo anterior possa ser devidamente mensurado. Após obter criar um diretório para as imagens finais, o mesmo será preenchido por dois *datasets* de vários rostos desenhados no estilo *anime*[7,8], e a partir das 26 mil imagens coletadas, é desenvolvido e executado um algoritmo que irá fazer o reconhecimento de bordas e, para cada imagem no diretório original, criar uma imagem do seu *outline* em um outro diretório com o mesmo nome de arquivo. A detecção de bordas, por ser um processo já amplamente estudado no campo da visão computacional [Marr e Hildreth 1980, Canny 1987], e de implementação robusta na API OpenCV [Laganière 2014], não têm seu desenvolvimento incluído sob o escopo do trabalho.

A rede VAEGAN pode então ser desenvolvida para que cada imagem amostrada em um diretório (por conveniência denominado diretório A) que será usado como entrada na rede G, será também amostrada uma imagem de mesmo nome em outro diretório

[7] Disponível em: <https://github.com/Mckinsey666/Anime-Face-Dataset> Acesso em junho de 2019.
[8] Disponível em: <http://www.nurs.or.jp/∼nagadomi/animeface-character-dataset/> Acesso em junho de 2019.

(denominado diretório B). No diretório A estarão as imagens dos *outlines*, enquanto B contém a imagem na sua forma final. A rede D da VAEGAN terá assim que comparar entre uma imagem retirada do diretório A que passou pelo processo de *encoder-decoder* da rede G e sua imagem equivalente retirada do diretório B, para assim decidir qual será a imagem "real".

O desenvolvimento da arquitetura foi dada de forma similar a aplicação CycleGAN [Zhu et al. 2017] do uso de GANs condicionais para tradução de imagens. A nova arquitetura desenvolvida encontra-se apresentada na Tabela 3, devido ao grande aumento na complexidade da arquitetura a técnica de *batch normalization* não será mais implementada, devido ao fato de isso resultaria em um volume de variáveis a ser computadas grande demais para o sistema utilizado no desenvolvimento.

Tabela 3 – Arquitetura da rede G (VAE).

Camada	**Operação**	**Dimensão de Saída**	**Ativação**
Entrada	Extração de valores RGB	128x128x3	Linear
Encode 1	Conv2D	128x128x64	LReLU
Encode 2	Conv2D	64x64x128	LReLU
Encode 3	Conv2D	32x32x256	LReLU
Encode 4	Conv2D	16x16x512	LReLU
Encode 5	Conv2D	8x8x512	LReLU
Encode 6	Conv2D	4x4x512	LReLU
Encode 7	Conv2D	2x2x512	LReLU
Latente	Conv2D	1x1x512	LReLU
Decode 1	TransConv2D	2x2x512	LReLU
Decode 2	TransConv2D	4x4x512	LReLU
Decode 3	TransConv2D	8x8x512	LReLU
Decode 4	TransConv2D	16x16x512	LReLU
Decode 5	TransConv2D	32x32x256	LReLU
Decode 6	TransConv2D	64x64x128	LReLU
Decode 7	TransConv2D	128x128x64	LReLU
Saída	FProp	128x128x3	TanH

Além da ausência da técnica de estabilização do treinamento, outra característica a ser notada da arquitetura desenvolvida é que a mesma recebe uma imagem, cria um vetor de características codificado, depois faz a decodificação do vetor para gerar outra imagem. Quando comparada a rede G anterior, a rede presente é mais complexa e recebe um sinal de entrada no mesmo formato do sinal de saída.

A rede D, conforme o apresentado na Tabela 4, também é mais complexa que sua implementação de GAN simples, no entanto, pela tarefa do discriminador ser mais simples quando comparada a da rede G, o aumento em complexidade é desproporcional de forma a favorecer a rede G. A implementação em código da VAEGAN é feita de forma idêntica a da GAN simples, porém com as alterações necessárias para as mudanças feitas na entrada,

Tabela 4 – Arquitetura da rede D.

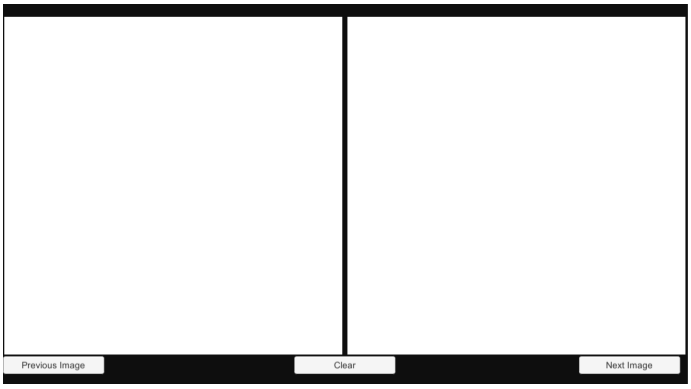
Camada	**Operação**	**Dimensão de Saída**	**Ativação**
Entrada	Extração de valores RGB	128x128x3	Linear
Encode 1	Conv2D	128x128x6	LReLU
Encode 2	Conv2D	128x128x64	LReLU
Encode 3	Conv2D	64x64x128	LReLU
Encode 4	Conv2D	32x32x256	LReLU
Encode 5	Conv2D	32x32x512	LReLU
Encode 6	Conv2D	32x32x1	LReLU
Saída	FProp	1	Sigmoid

de forma que faça amostragens de imagens de mesmo nome em diretórios distintos.

Para a VAEGAN, após os bancos de dados e arquitetura estarem em funcionamento, também é feito o treinamento ao longo do período de aproximadamente três semanas com 80 iterações, também armazenando os pesos periodicamente. Os pesos armazenados são então utilizados para o desenvolvimento de uma aplicação visual que permite, em tempo real, alterar a imagem de entrada da VAEGAN para obter uma saída traduzida em tempo real.

A aplicação é desenvolvida com uma funcionalidade simples de pincel e borracha visando atender um artista em potencial usando das funcionalidades do Unity Canvas, com a possibilidade de usar uma das imagens originais do banco de dados para a realização de testas. Os testes feitos com a VAEGAN também serão detalhados no Capítulo 5, e a implementação gráfica da aplicação pode ser vista na Figura 14.

Figura 14 – Aplicação de desenho para VAEGAN em tempo real.



Fonte: Elaborada pelo autor.

5 Resultados

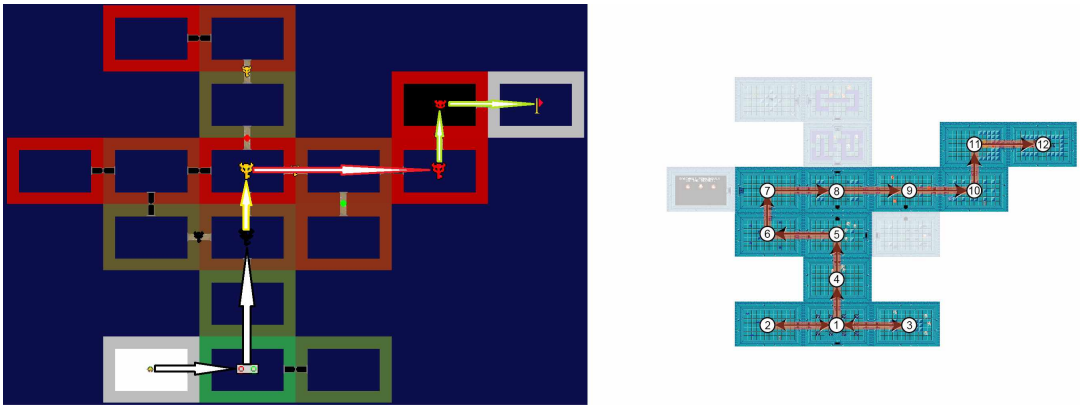
Assim como o capítulo anterior, os resultados serão avaliados em duas partes. A primeira etapa consiste detalhar a avaliação de resultados obtidos pela aplicação de algoritmos genéticos ao design de níveis automatizado, já a outra parte é realizada através do detalhamento dos resultados obtidos com os experimentos realizados na geração de imagens tanto com a arquitetura da GAN simples quanto a VAEGAN, assim como os resultados obtidos com sua implementação em tempo real.

5.1 Geração Genética de *Dungeons*

A implementação do algoritmo genético desenvolvida foi extremamente bem sucedida em realizar a tarefa proposta, podendo criar tanto uma distribuição aleatória de salas quanto encontrar a melhor disposição de conexões dentro de uma distribuição espacial definida por um designer, buscando sempre a menor não-linearidade e, devido a implementação genérica, deixando portas abertas para análise de outros fatores qualitativos.

Para parâmetro comparativo, foram usadas restrições espaciais seguindo o mesmo design que os *dungeons* do original *The Legend of Zelda*, e a não linearidade do caminho crítico comparada. É possível notar que, conforme o mostrado na Figura 15 que segue, o *dungeon* gerado não requer que o jogador passe mela mesma sala sequer uma vez quando seguindo o caminho crítico, quando comparado ao layout original que requer que o jogador passe pela sala (1) em três diferentes ocasiões.

Figura 15 – Comparação do caminho crítico gerado ao original.



Fonte: Elaborada pelo autor.

O fato de que o algoritmo genético foi capaz de, de forma consistente, gerar mais níveis como o apresentado é especialmente notável devido ao fato de que a presença de diferentes condições que aumentam o espaço de soluções possíveis como os diferentes

tipos de conexões, as chaves e as travas (que embora não são consideradas no algoritmo de adaptação, são posicionados de forma aleatória para atender o parâmetro qualitativo de variedade). O problema de grafos rotulados conectados é um problema notoriamente complexo, portanto o algoritmo genético conseguir soluções consistentes em um grande espaço de soluções possíveis, o que é evidenciado na Figura 16.

Figura 16 – Formas distintas de conectar grafos rotulados para n nós.

n	a(n)
0	1
1	1
2	1
3	4
4	38
5	728
6	26704
7	1866256
8	251548592
9	66296291072
10	34496488594816
11	35641657548953344
12	73354596206766622208
13	301272202649664088951808
14	2471648811030443735290891264
15	40527680937730480234609755344896
16	1328578958335783201008338986845427712

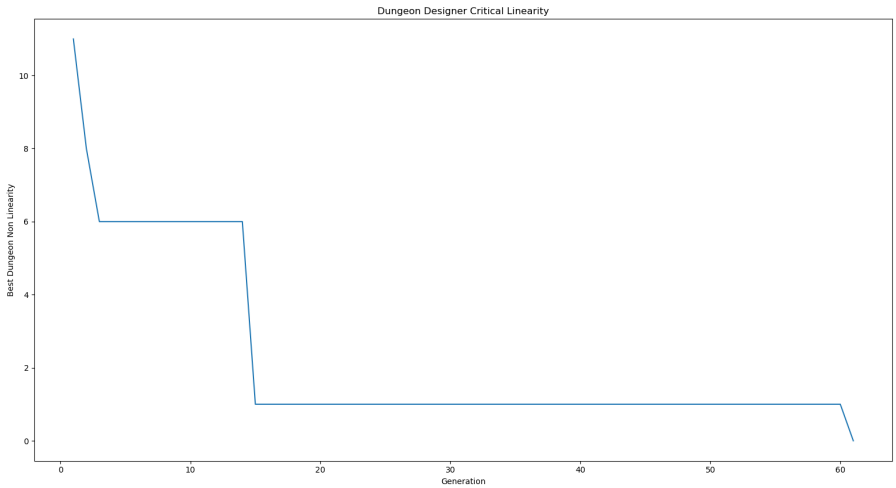
Fonte: Sequência A001187, OEIS[9].

Os resultados obtidos são notáveis por atenderem diversas demandas de design de níveis, oferecer uma solução rápida o bastante para atender as necessidades de geração procedural, e a implementação do elitismo se mostrou eficiente em garantir que a convergência ocorresse mesmo com uma população relativamente pequena de 50 *dungeons* por geração.

A população foi decidida através experimentação mostrar que era o ponto no qual o algoritmo convergia de forma consistente, porém não sofria com gargalos de performance que tornava a execução lenta. O número máximo de gerações foi fixado em 500 pelos mesmos motivos. Além disso, notadamente a implementação do JPS para algoritmo de busca foi bem sucedido devido ao fato de que um grande volume de grafos é analisado a cada geração, porém a tarefa era feita de forma veloz o bastante para aplicações em tempo real. A convergência gradual pode ser vista na Figura 17, onde o melhor indivíduo sempre será mantido de uma geração para a próxima de forma a evitar que o algoritmo perca boas soluções.

[9] Disponível em: <http://oeis.org/A001187/list> Acesso em Junho de 2019.

Figura 17 – Não-linearidade do melhor indivíduo por geração.



Fonte: Elaborada pelo autor.

A flexibilidade da implementação genérica também permitia, quando não utilizada as restrições espaciais, a geração de *dungeons* com n salas. É importante também ressaltar que o tempo necessário para alcançar soluções satisfatórias crescem de forma explosiva em relação ao número de salas desejadas. Quando comparada à geração automatizada de *dungeons* tal como é encontrada em *Bloodborne* (FROMSOFTWARE, 2015), a solução alcançada no trabalho é capaz de gerar disposições espaciais muito mais vastas e complexas.

5.2 *Sprites* e Modelos Generativos

Pela implementação e desenvolvimento feito através das GANs e VAEGANs serem significativamente distintos, a seção presente tratará de ambas separadamente. Pelo fato das VAEGANs serem um modelo que visa incrementar e resolver algumas dificuldades encontradas no modelo VAE e no modelo GAN, também é importante ressaltar as diferenças de resultados obtidos, principalmente de forma qualitativa.

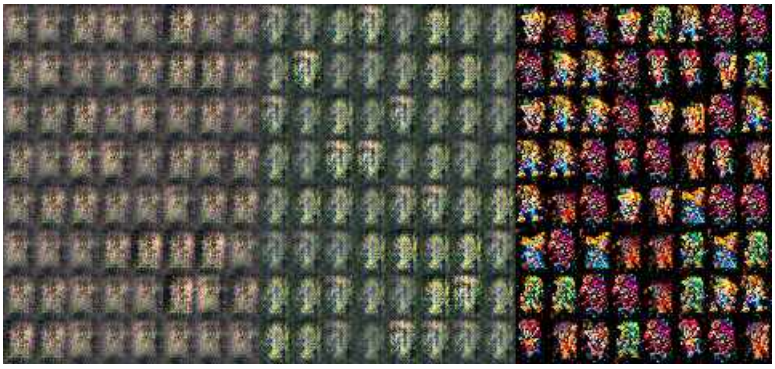
5.2.1 Geração de *sprites* por meio das GANs

Inicialmente, as funções de ativação usadas na arquitetura vista na Tabela 1 e 2 eram do tipo retificador linear (encontrado na literatura regularmente como $ReLU$) e a camada de saída possuía ativação do tipo Sigmoid, porém a medida que foram realizadas diferentes iterações de treinamento a arquitetura foi sofrendo alterações até chegarem no estado final que é apresentado conforme as tabelas mencionadas.

Com a rede estruturada e com o treinamento dado início, era amostrado uma saída de $G(z^{(i)})$ a cada 9 passos de treinamento. Após os primeiros nove passos de treinamento,

a imagem amostrada era praticamente ruído, porém como é possível notar na Figura 18 Após um extenso período de treinamento, a rede chegou em um ponto no qual G tornou-se incapaz de produzir melhoras visíveis, no entanto os resultados alcançados já começam a visivelmente se parecer com aqueles encontrados na Figura 12, portanto é possível ver que a GAN, embora não apresentando resultados muito fidedignos, já se mostram capazes de aproximar um determinado estilo, por mais abstrato que o mesmo seja.

Figura 18 – Resultados em 9, 18 e 306 passos.



Fonte: Elaborada pelo autor.

Houveram 2 problemas que foram atacados ao decorrer das futuras etapas de treinamento, sendo estes principalmente:

- Gradiente aproximando-se de 0.
- Excesso de ruído na saída.
- Repetições excessivas.

Uma das soluções encontradas na literatura [Isola et al. 2017] propõe que, para solucionar o problema do desaparecimento do gradiente, seja somado um espaço de ruído gaussiano a amostra real x. No entanto, a solução proposta não foi eficiente no caso em particular pois não foram percebidas alterações visíveis nos resultados em novos testes com ruído adicionado.

Uma hipótese para o porquê disso está no fato de que, pelo fato das imagens serem pequenas, a adição de um ruído gaussiano em *pixel-art* é diretamente antagônico à principal caraterística do estilo artístico, que é sua precisão a nível de pixel visto que é possível percebê-los a olho nu portanto a adição de ruído, por mais que pequena, já provocaria alterações muito significativas em x amostrado.

Outro conjunto de soluções testadas estão nas diferentes funções de ativação que foram desenvolvidas visando a melhoria de treinamento, principalmente das RNNs [Pascanu, Mikolov e Bengio 2013, Xu et al. 2015], porém foram adotadas em outros modelos também. Sendo estas a tangente hiperbólica na camada de saída, e a utilização de uma alteração

do retificador linear denominada de Leaky ReLU, ambas desenvolvidas para atacar o problema recorrente do desaparecimento do gradiente. A tangente hiperbólica é conhecida por melhorar a definição de imagens geradas pois valores negativos quando traduzidos para RGB são truncados em preto, o que em *pixel-art* é ainda mais benéfico, especialmente visto que a faixa da tangente hiperbólica vai de -1 a 1 enquanto a Sigmoid vai somente de 0 a 1, com todos os valores maiores que zero produzindo cor.

Figura 19 – Resultados após mais 90 iterações com as novas funções de ativação.

Fonte: Elaborada pelo autor.

As melhorias com as novas funções de ativação foram visíveis, conforme o visto na Figura 19. A redução de ruído sendo a principal melhoria observada nos novos resultados obtidos. Alguns problemas no entanto, persistiram, em especial o fato da rede G estagnar de forma precipitada.

Uma hipótese para explicar a estagnação da arquitetura é de que, também pela pequena dimensão das imagens, a rede D foi capaz de "memorizar"amostras reais $x^{(i)}$ e geradas $G(z^{(i)})$ de tal forma que $D(G(z^{(i)})) \approx 0$ e $D(x^{(i)}) \approx 1$, o que resultaria em uma perda do gradiente ∇_{θ_G}. A perda de gradiente pode ser demonstrada de forma trivial, pois G descende o gradiente:

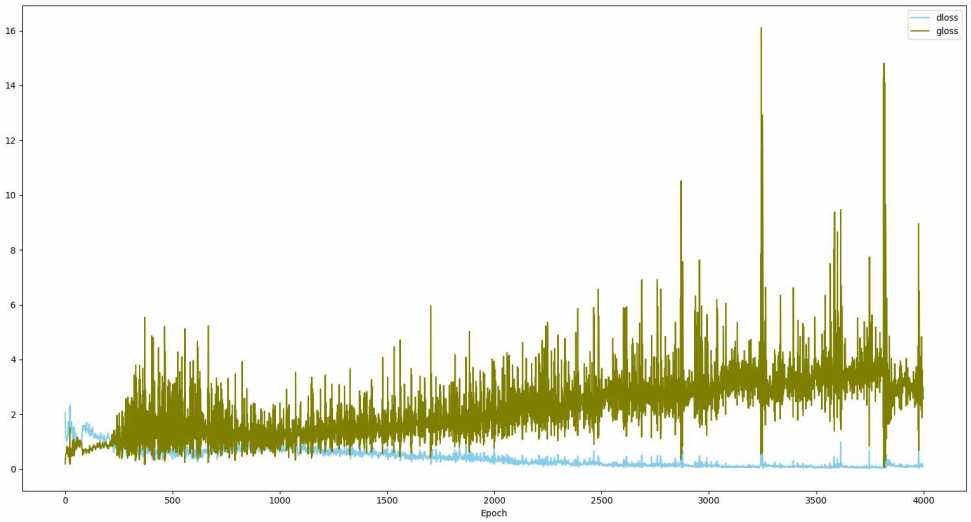
$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(G(z^{(i)})))]$$

E para $D(G(z^{(i)})) \approx 0$ e $D(x^{(i)}) \approx 1$, temos:

$$\begin{aligned} & \nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(G(z^{(i)})))] \\ & = \nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - 0)] \\ & = \nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} [\log(1)] \\ & = \nabla_{\theta_G} 0 \end{aligned} \tag{1}$$

Uma forte evidência de que o que foi demonstrado matematicamente no Desenvolvimento (1) de fato ocorreu no trabalho desenvolvido encontra-se na Figura 20, onde o Jogo do Minmaxing ocorre de fato até aproximadamente 300 passos de execução. A partir de então a rede D continua minimizando suas perdas de forma que G não se torna mais capaz de acompanhar as melhorias no desempenho de D, com suas perdas se tornando cada vez mais erráticas devido ao desaparecimento do gradiente.

Figura 20 – Perdas da rede G e D ao longo de quatro mil passos.



Fonte: Elaborada pelo autor.

Outro problema detectado é um problema mais subjetivo, que é conhecido na literatura como *mode collapse* [Srivastava et al. 2017], tratando-se este fenômeno das repetições observadas ao longo das amostras de $G(z)$. Se *mode collapse* é ou não um problema irá depender de sua intensidade, pois, quando ocorre, as imagens que são repetidas costumam ter uma maior qualidade devido ao fato de mais consistentemente enganarem a rede D, isto é, são mais consistentemente similares à amostras reais.

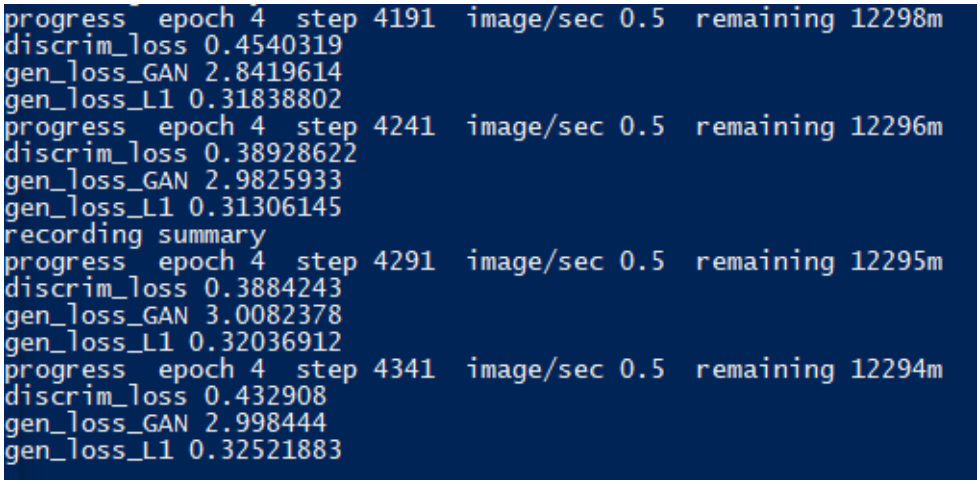
5.2.2 Tradução de Imagens com VAEGANs

A aplicação desenvolvida na plataforma Unity devido ao fato das imagens usadas para o treinamento possuírem aproximadamente sessenta vezes mais pixels que as usadas na GAN simples (que usava uma única imagem 20x28, contra duas imagens 128x128 usadas no treinamento da VAEGAN), e de uma quantidade de imagens aproximadamente quinze vezes maior (mais de 24 mil na VAEGAN contra 1700 da GAN simples), o treinamento da VAEGAN era um processo muito mais custoso e lento, o que se verifica na Figura 21. É possível ver que, apesar de ser uma arquitetura mais estável que manteve estabilidade

após milhares de passos, também é uma arquitetura mais lenta, que processa apenas meia imagem por segundo.

Graças a fatores como esses, os ajustes e experimentações não foram tão extensivas quanto nas GANs originais, no entanto os resultados foram melhores sem a necessidade de extensa experimentação e ajuste.

Figura 21 – Desempenho de treinamento da VAEGAN.



Fonte: Captura de tela feita pelo autor.

Algumas hipóteses para o porquê dos resultados terem sido melhores são as que seguem:

- Um conjunto de imagens reais x maior em dimensões e número de imagens, dificultando o trabalho da rede D, portanto o jogo não se torna desequilibrado em favor de D de forma tão rápida.
- Rede G se tornou muito mais complexa, fazendo a manipulação de uma imagem e não só de um simples vetor de ruído.
- Todas as melhorias de função de ativação elaborado para a GAN simples já foi herdado de forma direta na aplicação da VAEGAN desenvolvida.

As melhorias já se fazem evidentes de forma clara na Figura 22 onde, apesar da baixíssima riqueza de detalhes no rascunho feito pelo autor, na saída já é possível ver o que se assemelha a um rosto no estilo escolhido para o treinamento, já contando com a presença de cores, olhos, cabelo, rosto e boca. Os resultados são especialmente notáveis pelo fato de que se trata de uma implementação em tempo real, que usa somente da rede G foi reprogramada na forma de *shader* de forma a usar a GPU.

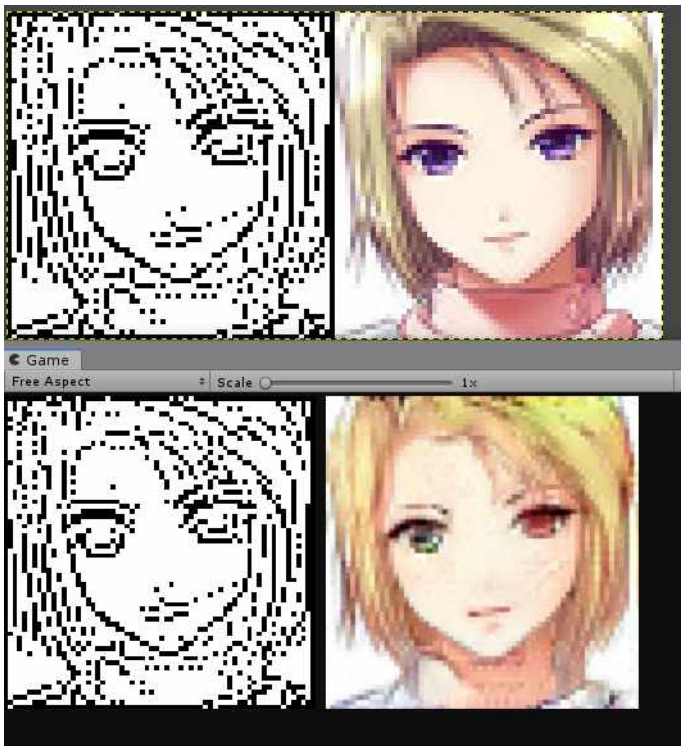
Para implementações em tempo real tal como a que foi desenvolvida, fazer um bom uso da GPU é fundamental pois caso contrário não seria uma ferramenta útil para um

artista devido a grande latência em tempos de resposta. Para a validação da saída da rede, também foi feita uma comparação entre pares de imagem reais com o par gerado pela rede em questão. Os testes também são feitos dentro da plataforma Unity usando da classe Texture2D que irá conter a imagem do *outline* retirada do banco de dados. A rede então irá produzir a saída da imagem do lado direito, que será comparada com a imagem final real.

Figura 22 – Saída da VAEGAN para rascunho feito pelo autor.

Fonte: Elaborada pelo autor.

Figura 23 – Comparação entre par real do banco de dados acima com par gerado abaixo.

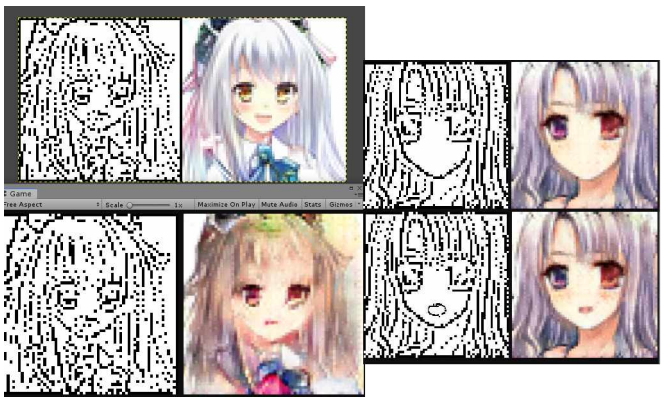


Fonte: Elaborada feita pelo autor.

Na Figura 23 é possível notar a precisão do modelo feito quando o *outline* é retirado do banco de dados, e nota-se que a qualidade da imagem é extremamente elevada, contando até mesmo com elementos de sombreamento bem definidos. Já na Figura 24, é possível observar como o modelo comporta-se em diferentes circunstâncias, na esquerda foi realizado

o mesmo teste da figura anterior, porém com uma imagem que não fazia parte do banco de dados e na direita foi realizado um teste onde modificações foram feitas ao *outline* original, de forma a adicionar uma boca ao desenho final.

Figura 24 – Diferentes testes realizados com o modelo.

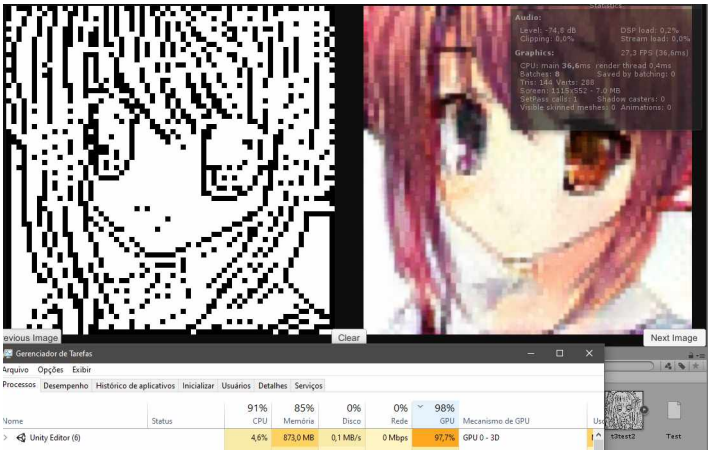


Fonte: Elaborada pelo autor.

Como pode ser feito acima, o modelo se comportou de forma adequada para ambas as situações, assim validando a utilidade da ferramenta para a criação e modificação de desenhos produzidos de forma a acelerar a criação dos mesmos. Algo recorrente que pode ser observado no modelo foi que para quase todas as amostras geradas, foi notada a presença do fenômeno da heterocromia ocular [Imesch, Wallow e Albert 1997].

Uma hipótese levantada sobre o porquê do fenômeno da heterocromia ocular ser tão prevalente nas saídas da rede amostradas é de que o fato da variação de cores ser muito grande ao longo das amostras reais do banco de dados, a rede pode convergir nas características mais consistentes tais como forma de olho, boca e nariz, com mais sucesso do que em detalhes finos tais como a cor dos olhos. Outras características mais consistentes como a cor da esclera ocular e pupila mantiveram-se constantes, assim como cor de pele e cabelo embora estas também apresentam variações significativas.

Figura 25 – Desempenho da aplicação em execução.



Fonte: Elaborada pelo autor.

No aspecto de desempenho, a implementação em tempo real desenvolvida também foi extremamente bem sucedida, capaz de manter, em média, entre 27 e 30 *frames* por segundo durante a execução, isto é, a rede é capaz de executar $Dec(Enc(x^{(i)}))$ até trinta vezes por segundo. Isso é possível pelo o que é mostrado na Figura 25, onde o uso da GPU, devido a implementação dos cálculos na forma de *shader* é sempre próximo de 100%. A técnica utilizada, conhecida como Compute Shading [Ni 2009], foi elaborada para usar do fato de que *shaders* geram programas de GPU para fazer computação numérica com paralelismo em massa.

6 Considerações finais

Ao longo da realização do trabalho, foram analisadas e testadas diversas técnicas para resolver diferentes problemas encontrados no processo de produção de mídia em 2D, seja ela um jogo ou uma animação ou até mesmo uma pintura. Para isso as soluções buscadas foram focadas principalmente no uso de algoritmos de inteligência artificial visando dar aos artistas ferramentas que tornariam a criação de arte 2D tão competitiva quanto a criação dos seus equivalentes tridimensionais.

É importante ressaltar o valor desse esforço em preservar a competitividade econômica e todo um subconjunto de arte que tem perdido espaço devido a, principalmente, fatores econômicos e de facilidade de produção. Esses fatores podem ser corrigidos com novas ferramentas, que tornam novamente mais nivelado o cenário artístico presente.

A pesquisa desenvolvida nos levou às fronteiras do que foi atualmente desenvolvido no campo da inteligência artificial no campo artístico, e até mesmo a elaboração de novos algoritmos.

Uma importante contribuição da pesquisa foi mostrar o quão poderoso é o algoritmo genético, e como suas implementações e reimplementações ainda podem ser aproveitadas de inúmeras maneiras distintas, mostrando seu constante potencial em ser usado para diferentes aplicações. No caso o produto final pode ser adaptado em uma excelente ferramenta para designers de níveis, dependendo apenas do critério de avaliação determinado à critério do mesmo.

Os modelos generativos também mostraram-se eficientes em conseguir abstrair um estilo específico, e além disso são capazes de criar amostras similares as amostras originais. A precisão, principalmente da GAN simples, deixa a desejar para aplicações que têm grandes requerimentos de precisão, no entanto a VAEGAN se mostrou extremamente apta para fazer a tradução de um rascunho para um desenho finalizado no estilo escolhido.

A aplicação em tempo real mostrou-se capaz de auxiliar, até mesmo alguém inepto, a produzir um desenho final que se assemelhasse aos desejados. Para a aplicação também é interessante ressaltar que a mesma só contava com os instrumentos mais básicos de edição de imagem, sendo capaz de somente desenhar e apagar um desenho. A possibilidade de incrementar as ferramentas de edição de imagem também mostra um grande potencial de melhoras na aplicação desenvolvida, podendo vir a se tornar uma ferramente excelente para artistas em potencial.

Os originais objetivos propostos foram concluídos em ambas as suas partes, o estudo de modelos generativos foi extenso, porém os resultados foram frutíferos, e ao final

do trabalho tanto a geração de mapas quanto a de *sprites* foi bem sucedida, concluindo assim os objetivos a serem alcançados.

Para trabalhos futuros, seria interessante trabalhar com novas arquiteturas, que fazem melhor uso da GPU, para alcançar melhores resultados com imagens maiores, pois no momento presente o projeto não é capaz de escalar de forma trivial, tornando limitado o escopo de problemas que o mesmo é capaz de resolver. Além disso, a possibilidade de adicionar novas ferramentas de edição de imagens também poderia ser estudada, possibilitando a criação de uma ferramenta mais robusta para ser utilizada por artistas que atuam no campo.

Também é possível, para os algoritmos genéticos, identificar aplicações similares, porém voltadas à geração de ambientes tridimensionais, pois o aumento de complexidade espacial dificulta ainda mais o trabalho de um designer de níveis.

Ao fim do trabalho é possível, com o uso das ferramentas criadas, automatizar parte significativa do processo de criação de insumos 2D para, no caso, o desenvolvimento de um jogo, e outras etapas do processo também podem ser estudadas para futuras implementações.

Referências

ARJOVSKY, M.; BOTTOU, L. Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862, 2017. Citado na página 39.

ARJOVSKY, M.; CHINTALA, S.; BOTTOU, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. Citado na página 39.

BANERJI, C. R.; MANSOUR, T.; SEVERINI, S. A notion of graph likelihood and an infinite monkey theorem. *Journal of Physics A: Mathematical and Theoretical*, IOP Publishing, v. 47, n. 3, p. 035101, 2013. Citado na página 51.

BARTLE, R. A. *Designing virtual worlds*. [S.l.]: New Riders, 2004. Citado na página 26.

BLEI, D. M.; KUCUKELBIR, A.; MCAULIFFE, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, Taylor & Francis, v. 112, n. 518, p. 859–877, 2017. Citado na página 42.

BOREL, É. La mécanique statique et l'irréversibilité. *J. Phys. Theor. Appl.*, v. 3, n. 1, p. 189–196, 1913. Citado na página 27.

BOTEA, A.; MÜLLER, M.; SCHAEFFER, J. Near optimal hierarchical path-finding. *Journal of game development*, Citeseer, v. 1, n. 1, p. 7–28, 2004. Citado na página 52.

BOTTOU, L. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nimes*, v. 91, n. 8, p. 12, 1991. Citado na página 35.

BYRNE, E. *Game level design*. [S.l.]: Charles River Media Boston, 2005. v. 6. Citado na página 26.

CALABRESE, D. *Unity 2D game development*. [S.l.]: Packt Publishing Ltd, 2014. Citado na página 47.

CANNY, J. A computational approach to edge detection. In: *Readings in computer vision*. [S.l.]: Elsevier, 1987. p. 184–203. Citado na página 56.

CANOSSA, A.; SMITH, G. Towards a procedural evaluation technique: Metrics for level design. In: *The 10th International Conference on the Foundations of Digital Games*. [S.l.: s.n.], 2015. p. 8. Citado na página 31.

CARLESS, S. *Gaming hacks*. [S.l.]: "O'Reilly Media, Inc.", 2004. Citado na página 28.

COHN, N. *The Visual Linguist Burnt City*. 2006. Disponível em: <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm>. Citado na página 19.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989. Citado na página 41.

DARWIN, C. *On the origin of species, 1859*. [S.l.]: Routledge, 2004. Citado na página 27.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002. Citado na página 28.

DOERSCH, C. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016. Citado na página 41.

FOLEY, J. D. et al. *Computer graphics: principles and practice*. [S.l.]: Addison-Wesley Professional, 1996. v. 12110. Citado na página 19.

GANDOMI, A. H.; ALAVI, A. H.; RYAN, C. *Handbook of genetic programming applications*. [S.l.]: Springer, 2015. Citado na página 22.

GARDNER, M. W.; DORLING, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, Elsevier, v. 32, n. 14-15, p. 2627–2636, 1998. Citado na página 41.

GATYS, L. A.; ECKER, A. S.; BETHGE, M. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015. Citado na página 45.

GOODFELLOW, I. et al. Generative adversarial nets. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2014. p. 2672–2680. Citado na página 33.

GÖRING, F. Short proof of menger's theorem. *Discrete Mathematics*, Elsevier, v. 219, n. 1-3, p. 295–296, 2000. Citado na página 25.

GROSS, J. L.; YELLEN, J. *Handbook of graph theory*. [S.l.]: CRC press, 2004. Citado na página 25.

HARABOR, D. D.; GRASTIEN, A. Online graph pruning for pathfinding on grid maps. In: *Twenty-Fifth AAAI Conference on Artificial Intelligence*. [S.l.: s.n.], 2011. Citado na página 52.

HARABOR, D. D.; GRASTIEN, A. The jps pathfinding system. In: *SOCS*. [S.l.: s.n.], 2012. Citado na página 52.

HARABOR, D. D.; GRASTIEN, A. Improving jump point search. In: *Twenty-Fourth International Conference on Automated Planning and Scheduling*. [S.l.: s.n.], 2014. Citado na página 52.

HECHT-NIELSEN, R. Theory of the backpropagation neural network. In: *Neural networks for perception*. [S.l.]: Elsevier, 1992. p. 65–93. Citado na página 35.

HECKER, C. *Perspective Texture Mapping*. 1996. Disponível em: <http://www.gamers.org/dEngine/quake/papers/checker_texmap.html>. Citado na página 20.

HOCKING, J. *Unity in action: Multiplatform game development in C# with Unity 5*. [S.l.]: Manning Publications Shelter Island, NY, 2015. Citado na página 47.

HONOUR, H.; FLEMING, J. *A world history of art*. [S.l.]: Laurence King Publishing, 2005. Citado na página 19.

HORN, B. et al. A comparative evaluation of procedural level generators in the mario ai framework. In: SOCIETY FOR THE ADVANCEMENT OF THE SCIENCE OF DIGITAL GAMES. *Foundations of Digital Games 2014*. [S.l.], 2014. Citado na página 30.

HUANG, X. et al. Multimodal unsupervised image-to-image translation. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 172–189. Citado na página 46.

IMESCH, P. D.; WALLOW, I. H.; ALBERT, D. M. The color of the human eye: a review of morphologic correlates and of some conditions that affect iridial pigmentation. *Survey of ophthalmology*, Elsevier, v. 41, p. S117–S123, 1997. Citado na página 67.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Citado na página 55.

ISOLA, P. et al. Image-to-image translation with conditional adversarial networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1125–1134. Citado na página 62.

JEBARA, T. *Machine learning: discriminative and generative*. [S.l.]: Springer Science & Business Media, 2012. v. 755. Citado na página 33.

JORDAN, M. I.; MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science*, American Association for the Advancement of Science, v. 349, n. 6245, p. 255–260, 2015. Citado na página 33.

JULIANI, A. et al. Unity: A general platform for intelligent agents. *CoRR*, abs/1809.02627, 2018. Disponível em: <http://arxiv.org/abs/1809.02627>. Citado na página 47.

KANTROWITZ, A.; BREW, A.; FAVA, M. Thinking through drawing: Practice into knowledge. In: *Proceedings of an interdisciplinary symposium on drawing, cognition and education*. [S.l.: s.n.], 2011. p. 123–125. Citado na página 19.

KENT, S. L. *The Ultimate History of Video Games: Volume Two: from Pong to Pokemon and beyond... the story behind the craze that touched our li ves and changed the world*. [S.l.]: Three Rivers Press, 2010. Citado na página 53.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Citado na página 35.

KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. Citado na página 40.

KLEINER, F. S. *Gardner's art through the ages: The western perspective*. [S.l.]: Cengage Learning, 2016. v. 1. Citado na página 19.

KRAMER, O. *Genetic algorithm essentials*. [S.l.]: Springer, 2017. v. 679. Citado na página 50.

KUPERBERG, M. *Guide to Computer Animation: for tv, games, multimedia and web*. [S.l.]: Routledge, 2012. Citado na página 20.

LAGANIÈRE, R. *OpenCV Computer Vision Application Programming Cookbook Second Edition*. [S.l.]: Packt Publishing Ltd, 2014. Citado na página 56.

LARMAN, C. Object-oriented analysis and design. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, p. 10–11, 2002. Citado na página 50.

LARSEN, A. B. L. et al. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015. Citado na página 43.

LE, Q. V. et al. On optimization methods for deep learning. In: OMNIPRESS. *Proceedings of the 28th International Conference on International Conference on Machine Learning*. [S.l.], 2011. p. 265–272. Citado na página 35.

LEACHTENAUER, J. C. et al. General image-quality equation: Giqe. *Applied optics*, Optical Society of America, v. 36, n. 32, p. 8322–8328, 1997. Citado na página 43.

LESTER, N. A. Disney's the princess and the frog: The pride, the pressure, and the politics of being a first. *The Journal of American Culture*, Blackwell Publishing Ltd., v. 33, n. 4, p. 294, 2010. Citado na página 20.

LIOU, C.-Y. et al. Autoencoder for words. *Neurocomputing*, Elsevier, v. 139, p. 84–96, 2014. Citado na página 40.

LIOU, C.-Y.; HUANG, J.-C.; YANG, W.-C. Modeling word perception using the elman network. *Neurocomputing*, Elsevier, v. 71, n. 16-18, p. 3150–3157, 2008. Citado na página 40.

MARR, D.; HILDRETH, E. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, The Royal Society London, v. 207, n. 1167, p. 187–217, 1980. Citado na página 56.

MEIJER, E.; BECKMAN, B.; BIERMAN, G. Linq: reconciling object, relations and xml in the. net framework. In: ACM. *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*. [S.l.], 2006. p. 706–706. Citado na página 50.

MESCHEDER, L.; NOWOZIN, S.; GEIGER, A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In: JMLR. ORG. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. [S.l.], 2017. p. 2391–2400. Citado na página 43.

MIKHAIL, E. M.; BETHEL, J. S.; MCGLONE, J. C. Introduction to modern photogrammetry. *New York*, 2001. Citado na página 19.

MILLER, R. A. *The japanese language*. [S.l.]: University of Chicago Press Chicago, 1967. Citado na página 51.

MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998. Citado na página 27.

MITCHELL, R. *Web Scraping with Python: Collecting More Data from the Modern Web*. [S.l.]: "O'Reilly Media, Inc.", 2018. Citado na página 54.

NASH, J. Non-cooperative games. *Annals of mathematics*, JSTOR, p. 286–295, 1951. Citado na página 34.

NG, A. Y.; JORDAN, M. I. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2002. p. 841–848. Citado na página 33.

NI, T. Direct compute: Bring gpu computing to the mainstream. In: *GPU Technology Conference*. [S.l.: s.n.], 2009. p. 23. Citado na página 68.

OXLAND, K. *Gameplay and design*. [S.l.]: Pearson Education, 2004. Citado na página 26.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. In: *International conference on machine learning*. [S.l.: s.n.], 2013. p. 1310–1318. Citado na página 62.

RADFORD, A.; METZ, L.; CHINTALA, S. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. Citado na página 54.

ROSENBLATT, F. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. [S.l.], 1961. Citado na página 41.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. *Learning internal representations by error propagation*. [S.l.], 1985. Citado na página 41.

SALIMANS, T. et al. Improved techniques for training gans. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2016. p. 2234–2242. Citado na página 39.

SCHILDT, H. *C# 4.0: The complete reference*. [S.l.]: Tata McGraw-Hill Education, 2010. Citado na página 50.

SHAKER, N.; TOGELIUS, J.; NELSON, M. J. *Procedural content generation in games*. [S.l.]: Springer, 2016. Citado na página 22.

SILVA, N. B. et al. A graph-based friend recommendation system using genetic algorithm. In: IEEE. *IEEE Congress on Evolutionary Computation*. [S.l.], 2010. p. 1–7. Citado na página 31.

SNELL, J. et al. Learning to generate images with perceptual similarity metrics. In: IEEE. *2017 IEEE International Conference on Image Processing (ICIP)*. [S.l.], 2017. p. 4277–4281. Citado na página 45.

SRIVASTAVA, A. et al. Veegan: Reducing mode collapse in gans using implicit variational learning. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2017. p. 3308–3318. Citado na página 64.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, MIT Press, v. 10, n. 2, p. 99–127, 2002. Citado na página 31.

STOUT, M. *Learning From The Masters: Level Design In The Legend Of Zelda*. 2012. Disponível em: <https://www.gamasutra.com/view/feature/134949/learning_from_the_masters_level_.php>. Citado na página 28.

SUMNER, R. W. et al. Mesh-based inverse kinematics. In: ACM. *ACM transactions on graphics (TOG)*. [S.l.], 2005. v. 24, n. 3, p. 488–495. Citado na página 19.

SYLVESTER, J. J. *Chemistry and algebra*. [S.l.]: Nature Publishing Group, 1878. Citado na página 25.

SYLVESTER, J. J. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, with three appendices. *American Journal of Mathematics*, JSTOR, v. 1, n. 1, p. 64–104, 1878. Citado na página 25.

SZYPERSKI, C.; GRUNTZ, D.; MURER, S. *Component software: beyond object-oriented programming*. [S.l.]: Pearson Education, 2002. Citado na página 48.

TOGELIUS, J. et al. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 3, n. 3, p. 172–186, 2011. Citado na página 22.

TVERSKY, B. Visualizing thought. In: *Handbook of human centric visualization*. [S.l.]: Springer, 2014. p. 3–40. Citado na página 19.

WANG, X.; GUPTA, A. Generative image modeling using style and structure adversarial networks. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2016. p. 318–335. Citado na página 22.

WANG, Z.; BOVIK, A. C. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE signal processing magazine*, IEEE, v. 26, n. 1, p. 98–117, 2009. Citado na página 45.

WEISSTEIN, E. W. Normal distribution. Wolfram Research, Inc., 2002. Citado na página 34.

XU, B. et al. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. Citado na página 62.

ZHAO, S.; SONG, J.; ERMON, S. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017. Citado na página 43.

ZHU, J.-Y. et al. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2017. p. 2223–2232. Citado na página 57.