UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA

ESCOLA SUPERIOR DE TECNOLOGIA

ENGENHARIA DE COMPUTAÇÃO

DANIEL AKIO CHEN

NEURO-EVOLUÇÃO APLICADA NA CONCLUSÃO DE JOGOS DE PLATAFORMA

Manaus

2022

DANIEL AKIO CHEN

NEURO-EVOLUÇÃO APLICADA NA CONCLUSÃO DE JOGOS DE PLATAFORMA

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheiro de Computação.

Orientador(a): Prof. Ricardo Rios Monteiro do Carmo

Manaus

2022

Universidade do Estado do Amazonas - UEA

Escola Superior de Tecnologia - EST

Reitor:

André Luiz Nunes Zogahib

Vice-Reitor:

Katia do Nascimento Couceiro

Diretor da Escola Superior de Tecnologia:

Ingrid Sammyne Gadelha

Coordenador do Curso de Engenharia de Computação:

Áurea Hileia da Silva Melo

Coordenador da Disciplina Projeto Final:

Polianny Almeida Lima

Banca Avaliadora composta por: *Data da Defesa: 27/05/2022.*

Prof. Ricardo Rios Monteiro do Carmo, Mestre*(Orientador)*

Prof. Elloá Barreto Guedes Costa, Doutora

Prof. Fábio Santos da Silva, Doutor

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

C518nn Chen, Daniel Akio
Neuro-evulução aplicada na conclusão de jogos de plataforma / Daniel Akio Chen. Manaus : [s.n], 2022.
82 f.: color.; 30 cm.

TCC - Graduação em Engenharia de Computação - Universidade do Estado do Amazonas, Manaus, 2022.
Inclui bibliografia
Orientador: Carmo, Ricado Rios Monteiro do
Coorientador: Costa, Elloá Barreto Guedes e Silva, Fábio Santos da

1. Inteligência Artificial. 2. Neuro-evulução. 3. Jogos. I. Carmo, Ricado Rios Monteiro do (Orient.). II. Costa, Elloá Barreto Guedes (Coorient.). III. Silva, Fábio Santos da (Coorient.). IV. Universidade do Estado do Amazonas. V. Neuro-evulução aplicada na conclusão de jogos de plataforma

Elaborado por Jeane Macelino Galves - CRB-11/463

AMAZONAS
GOVERNO DO ESTADO

FOLHA DE APROVAÇÃO

“Neuro-evolução aplicada na conclusão de jogos de plataforma”

DANIEL AKIO CHEN

Trabalho de Conclusão do Curso de Engenharia da Computação defendido e aprovado pela banca avaliadora constituída pelos professores:

Prof. Ricardo Rios Monteiro do Carmo, Mestre.
UNIVERSIDADE DO ESTADO DO AMAZONAS

Presidente

Prof. Elloá Barreto Guedes Costa, Doutora.
UNIVERSIDADE DO ESTADO DO AMAZONAS

Arguidora

Prof. Fábio Santos da Silva, Doutor.
UNIVERSIDADE DO ESTADO DO AMAZONAS

Arguidor

Manaus, 27 de maio de 2022.

Resumo

Com o crescente número de pessoas que consomem jogos eletrônicos é inevitável que a indústria de jogos eletrônicos também cresça. Com o aumento de acesso aos jogos eletrônicos, diversos profissionais de saúde tem se favorecido deles como ferramentas de apoio ao tratamento de pacientes. Contudo, muitos jogos ainda não são utilizados nesses procedimentos, porque os profissionais de saúde não contam com avaliações que ofereçam parâmetros para decidir se o nível de dificuldade de um determinado jogo está adequado ao perfil do paciente em tratamento. Esse trabalho apresenta um método de neuro-evolução de jogos, uma maneira de construir redes neurais para resolução e avaliação de jogos, além dos passos utilizados para construir tal ferramenta.

Palavras Chave: Inteligência Artificial. Neuro-evolução. Jogos.

Abstract

Due to the growing number of people consuming electronic games it is inevitable the gaming industry will grow as well. With the increased access to video games, many healthcare professionals have taken advantage of them as tools to support patient care. However, many games are still not used in these procedures because health professionals do not have assessments that provide parameters for deciding whether the level of difficulty of a particular game is appropriate to the profile of the patient being treated. This work presents a game neuro-evolution method, a way to build neural networks for solving and evaluating games, and the steps used to build such a tool.

Key-words: Artificial Intelligence. Neuro-evolution. Games.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

INTRODUÇÃO

Há alguns anos, jogos eletrônicos têm se tornado uma importante ferramenta educacional e terapêutica. Pelo lado da educação, eles podem estimular a coordenação, melhorar reflexos, potencializar ensino de idiomas, gerar interesse em assuntos diversos etc. (EXAME, 2020). Pelo lado medicinal, eles são capazes de ajudar a melhorar o desempenho cognitivo e aprimorar os reflexos de crianças, adolescentes e até mesmo pessoas com mais de 50 anos (LOPEZ-SAMANIEGO et al., 2014). Entretanto, a gama de jogos que são efetivamente utilizados para esses fins é limitada a alguns gêneros, como, por exemplo, realidade virtual e ritmo (CARDOSO; LANDENBERGER; ARGIMON, 2017).

Jogos que exigem mais atenção e reflexos, como os do gênero plataforma, normalmente não são utilizados pela educação ou medicina pelo fato de demandarem muito mais tempo e habilidade do usuário (BALLARD CLIVE G.; CORBETT, 2010). Para tentar solucionar este problema de acessibilidade aos jogos, o mercado de jogos criou um sistema de avaliação de mídia. Nesse sistema, jogadores que já experienciaram um determinado jogo podem relatar o nível de dificuldade, custo-benefício, *design* etc., que foram experienciados. Dessa maneira, o avaliador pode fornecer parâmetros para o leitor decidir se o nível de dificuldade de determinado jogo está adequado para seu gosto ou necessidade.

No sistema referido é necessário que um jogador humano experiencie o jogo para que uma avaliação possa ser feita. Porém, e se pudéssemos trocar o jogador humano por um jogador-máquina? Na área de Computação, não é difícil encontrar aplicações de Inteligência Artificial

e Aprendizado de Máquina em jogos. Elas podem ser encontradas tanto na área de geração de conteúdo (RISI et al., 2016) quanto no campo de customização (PEDERSEN C.; TOGELIUS, 2010). Contudo, uma outra área na qual também são utilizadas é na de seleção de ações, ou jogatina automática. Para isso, uma das maneiras mais utilizadas é o uso de Neuro-evolução, um método de criação de Redes Neurais que permite o aprendizado do sistema através de evolução.

Nesse trabalho, uma abordagem de Inteligência Artificial utilizando um algoritmo de Neuro-evolução será aplicado no jogo *Super Mario Bros.* Esse algoritmo deve ser capaz de aprender e criar redes neurais que sejam capazes de jogar o jogo autonomamente. Depois de criada uma rede neural que esteja apta a isso, uma avaliação será feita para a coleta de alguns dados importantes, como, por exemplo, números de botões pressionados, quantidade de tempo tomado etc. Essas variáveis serão utilizadas para quantificar uma dificuldade para o jogo ou fase. Por fim, esse nível de dificuldade adquirido pode ser utilizado por profissionais de saúde para avaliar a dificuldade de um jogo e, a partir disso, decidir se esse jogo está adequado ou não para ser aplicado ao tratamento de um paciente.

Esse trabalho está dividido em sete capítulos, incluindo a introdução. Nas próximas seções da introdução serão apresentadas as justificativas para o trabalho, os objetivos gerais e específicos, a metodologia usada para o desenvolvimento do trabalho, os materiais necessários e o cronograma de atividades. No Capítulo 2 é apresentada uma revisão da literatura. O Capítulo 3 apresenta uma análise de trabalhos relacionados. O Capítulo 4 apresenta algoritmos possíveis de Neuro-evolução, incluindo o algoritmo principal a ser utilizado. O Capítulo 5 apresenta a arquitetura a ser implementada. O Capítulo 6 cobre os resultados obtidos em tal experimento. E, por último, o Capítulo 7 trata das considerações finais.

1.1 JUSTIFICATIVA

Desde a Terceira Revolução Industrial, mais conhecida como Revolução Técnico-Científica (CFA, 2019), iniciada em meados do século XX, tem havido um aumento da busca por aparelhos eletrônicos. Isso também é verdade em relação aos jogos eletrônicos, que há alguns anos têm

se tornado uma importante ferramenta de entretenimento e educação (MATOS; LIMA, 2015). Em 2020, aproximadamente 73,4% dos brasileiros afirmaram ter acesso a jogos eletrônicos para fins de entretenimento, correspondendo a um aumento de 7,1% em relação ao ano anterior (EXAME, 2020). Jogos eletrônicos também têm sido utilizados como ferramenta educacional para estimular socialização e colaboração, desenvolver autonomia, auxiliar no desenvolvimento do pensamento crítico e analítico, gerar interesse no conteúdo explorado no jogo, potencializar o ensino de competências socio-emocionais e cognitivas (EXAME, 2020; SIENA et al., 2018).

Devido a essa demanda, empresas têm se utilizado de várias estratégias, como, por exemplo, tradução para outros idiomas (SOUZA, 2011) e divulgação em vários meios de comunicação (redes sociais, serviços de *streaming*, mensagens eletrônicas, propaganda em sítios Web etc.) para popularizar o acesso a jogos eletrônicos. Há também um esforço de popularização de jogos no que diz respeito a pessoas com deficiência, como, por exemplo, limitações visuais, de fala, de movimento e cognitiva (SOLARI, 2011). Por exemplo, Microsoft®, proprietária do console XBOX® (MICROSOFT, 2021a), criou um controle adaptativo configurável que se propõe a suprir necessidades de pessoas com mobilidade limitada (MICROSOFT, 2021b). Ferramentas como essa, além de viabilizar o acesso de pessoas com deficiência a jogos, também são aproveitadas por profissionais de saúde para auxiliar no tratamento de determinados pacientes.

Em um experimento, observou-se que a utilização de jogos eletrônicos ajudou a melhorar o desempenho cognitivo de pessoas com mais de 50 anos (ORDONEZ TIAGO NASCIMENTO; BORGES, 2017). Em outro experimento, os autores mostraram que a reabilitação cognitiva e a melhora dos reflexos dessa faixa etária também é possível (LOPEZ-SAMANIEGO et al., 2014). Além disso, jogos eletrônicos têm se mostrado eficazes para auxiliar o desenvolvimento da atenção e memória de crianças, adolescentes e jovens adultos (BARROSO et al., 2019). Jogos do gênero de realidade virtual, de ritmo ou de *fitness* têm sido utilizados para esse fim (CARDOSO; LANDENBERGER; ARGIMON, 2017; BALLARD CLIVE G.; CORBETT, 2010; SOARES et al., 2017). Entretanto, jogos do gênero plataforma, como, por exemplo, *Super Mario Bros.* e Sonic Adventure (WIKIPEDIA, 2021a), não são utilizados em terapias pois exigem coordenação motora complexa e uma resposta rápida por parte do jogador.

Para tentar estimular a acessibilidade e contornar o problema da dificuldade de coordenação e reflexo, uma das soluções que o mercado encontrou foi criar um sistema de avaliação de jogos, baseado no enredo do jogo, jogabilidade e dificuldade (APONTE; LEVIEUX; NATKIN, 2009; UNIVERSITY, 2017). Dessa maneira, terapeutas e profissionais de saúde podem avaliar se aquele determinado produto é coerente ou não para o tratamento de seus pacientes. O processo de avaliar um jogo é direto: depois do avaliador experienciar o jogo por inteiro, ele deve observar alguns aspectos, como *design*, repetibilidade, tática-sorte, dificuldade e custo-benefício para que, assim, ele possa criar uma avaliação sobre o jogo (DUCOSIM, 2020; PEGI, 2020).

Em Computação, interagir com os jogos utilizando Inteligência Artificial é um tema bem explorado (STANLEY; MIIKKULAINEN, 2004; FOGEL et al., 2004; PARKER; BRYANT, 2008; ZHU, 2020). Em alguns desses trabalhos, os autores utilizaram Aprendizado de Máquina para a criação de agentes autônomos para esses jogos (BERNER et al., 2019). Em outros, eles empregaram essa Inteligência Artificial para a criação de inimigos formidáveis (FOGEL et al., 2004). Uma maneira de criar essa Inteligencia Artificial que adiciona elementos ao jogo é a utilização de Algoritmos Genéticos e de Neuro-evolução.

Neuro-evolução, inspirada na evolução biológica de sistemas nervosos, é uma técnica que aplica métodos evolucionários na construção de redes neurais (LEHMAN; MIIKKULAINEN, 2013). Com tal rede neural, é possível encontrar soluções para problemas complexos e, consequentemente, inferir informações relevantes com base nos resultados encontrados. Tal técnica já foi testada em jogos, por exemplo, ao se jogar autonomamente Quake 2 (PARKER; BRYANT, 2008) e Ms. Pacman (SCHRUM; MIIKKULAINEN, 2014).

Nesse trabalho, optou-se por utilizar Neuro-Evolução para automatizar o processo de conclusão de jogos, para que, ao final, a ferramenta possa criar pontuações de dificuldade com base nos elementos do jogo, como, por exemplo, número de inimigos, quantidade de botões pressionados, distância até o objetivo etc. Com essas informações, o profissional de saúde pode utilizar esses dados para aferir se o nível de dificuldade de um determinado jogo é adequado ou não para o tratamento de um determinado paciente.

1.2 OBJETIVOS

1.2.1 GERAL

Esse trabalho tem por objetivo propor uma ferramenta que auxilia profissionais de saúde a avaliarem e decidirem sobre a adequabilidade de jogos no auxílio do tratamento de pacientes. Esse sistema toma como parâmetros jogabilidade e dificuldade para tomar decisões.

1.2.2 ESPECÍFICOS

- Analisar e investigar os algoritmos genéticos mais eficientes para a criação de redes neurais na área de Inteligência Artificial e Neuro-evolução.
- Implementar, treinar e validar o algoritmo escolhido, em um contexto realístico.
- Elaborar um critério que sintetize a pontuação indicativa da dificuldade de um certo jogo.

1.3 METODOLOGIA

Para o desenvolvimento desse trabalho, serão investigados alguns algoritmos genéticos para identificar métodos de codificação e de reconhecimento de padrões. Nesses métodos serão avaliados a complexidade, eficiência e tipos de variáveis possíveis. Uma vez definido o algoritmo mais apropriado para abordar os desafios associados ao trabalho, o algoritmo será implementado.

Utilizando de emulação, o algoritmo será treinado autonomamente. Nessa fase, serão investigadas as variáveis que melhor otimizem esse sistema e os principais padrões apresentados pela emulação.

A emulação será executada até que atinja a condição de aceitação, que é a finalização do nível do jogo escolhido. Quando isso ocorrer, o código será validado para verificar se os requisitos são atendidos.

Na fase seguinte uma interface será criada para interação com o usuário final.

Por fim, o trabalho monográfico será redigido.

1.4 MATERIAIS

- Computador.
- Ferramenta de emulação de jogos: Útil para emular jogos e treinar o algoritmo genético implementado.
- Linguagem Lua: será a principal linguagem utilizada no projeto pelo fato de ser muito estudada na utilização de inteligência artificial e afins.
- Jogo a ser emulado: Super Mario Bros. Ele foi o escolhido pelo fato de ter uma jogabilidade simples, porém desafiadora, e que exige reflexos do usuário.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Para desenvolver os objetivos desse trabalho, esse capítulo introduzirá uma revisão de literatura que servirá de base para a arquitetura a ser criada. As seções seguintes entram em detalhes sobre conceitos de Inteligência Artificial, Aprendizado de Máquina, Redes Neurais, Computação Evolucionária e, por fim, Neuro-Evolução.

2.1 INTELIGÊNCIA ARTIFICIAL

De acordo com LUGER (2004), Inteligência Artificial (*Artificial intelligence*, em inglês — IA) pode ser definida como ramo da Ciência da Computação que se preocupa com a criação de comportamentos inteligentes nos computadores. A definição de inteligência e comportamentos inteligentes não é muito bem clara pelo fato de ter ligação com a inteligência humana, área que possui poucas certezas e muitos questionamentos filosóficos. Contudo, sabe-se que a IA tenta emular os comportamentos humanos e racionais (RUSSELL; NORVIG; CANNY, 2003). Para realizar isso, ela se utiliza de técnicas estatísticas ou lógicas para fazer essas escolhas, assim, cumprindo um certo objetivo.

Esse campo da ciência está presente em diversas áreas do mundo moderno: Visão Computacional (HUANG, 1996) na forma de processamento de imagens e vídeos, reconhecimento de objetos e análise facial; Audição Computacional (WANG, 2010), como o reconhecimento de fala; Processamento de Linguagem Natural (KONGTHON et al., 2009), por exemplo, bate-papos

com robôs e tradução automático de textos; Aprendizado de Máquina (RUSSELL; NORVIG; CANNY, 2003), contendo sistemas de recomendação, sistemas de pesquisa, entre outros.

2.2 APRENDIZADO DE MÁQUINA

Aprendizado de Máquina (AM) é uma forma de Inteligência Artificial. Ele permite que um sistema consiga aprender métodos ou comportamentos através de dados ao invés de programação direta no código interno (HURWITZ; KIRSCH, 2018b).

Essa técnica se utiliza de uma variedade de algoritmos para iterativamente aprender com os dados que recebe para melhorar, descrever e predizer as saídas. Sítios Web, como, por exemplo, *Youtube* e *Netflix*, coletam os dados de navegação do usuário para que eles sejam inseridos em um modelo de AM (CAZELLA; NUNES; REATEGUI, 2010). Esse modelo, então, utilizando dados, tenta aprender uma lógica fundamental: Que conteúdo o usuário iria querer, baseado naquilo que ele assistiu?

Todos os algoritmos de AM seguem o mesmo princípio. Primeiro, o algoritmo recebe dados que são oferecidos à ele. Durante o treinamento, o algoritmo faz associações e encontra padrões nos dados. Devido à complexidade e à quantidade de dados, os padrões e conexões poderiam ser perdidas se esse trabalho fosse feito manualmente. Dessa maneira, devido às associações, as respostas geradas por esse algoritmo serão mais precisas. O resultado final de um aprendizado de máquina é um "modelo de aprendizado de máquina" (HURWITZ; KIRSCH, 2018a). Após finalizado o treinamento, se o usuário prover um dado ao modelo, o algoritmo irá produzir um resultado que pode ser relevante ou não para o problema. No caso de recomendações do *Netflix*, por exemplo, se inserirmos uma série ou filme, o modelo treinado ofereceria como resposta uma outra série ou filme que o usuário possa ter interesse.

Existem três categorias principais de Aprendizado de Máquina, dependendo do problema apresentado: aprendizado supervisionado, aprendizado não-supervisionado e aprendizado por reforço.

2.2.1 APRENDIZADO SUPERVISIONADO

Aprendizado Supervisionado, também conhecido como Aprendizado de Máquina Supervisionado, é definido pelo conhecimento prévio de características da base de dados pelo usuário-professor. Em outras palavras, essa definição significa que o conjunto de dados que o algoritmo trabalhará contêm informações, chamadas em inglês de *labels*, que definem um significado aos dados (MARSLAND, 2011). Para exemplificar, se um conjunto de dados possuir milhões de fotografias de animais, os *labels* seriam as informações inclusas do tipo de animal presente em cada foto. Com essa informação seria possível criar um modelo de aprendizado de máquina que consegue distinguir animais nas imagens, identificando assim centenas ou milhares de diferentes espécies animais (HURWITZ; KIRSCH, 2018a).

Podem existir dois tipos de *labels* dentro do conjunto de dados, a informação contínua (e.g. peso, altura, comprimento) e a informação em um conjunto finito (e.g. espécies de animais) (IBMCLOUD, 2020a). Quando se tem dados contínuos, diz-se que se tem um problema regressão. Analogamente, em dados finitos tem-se um problema de classificação. Como exemplo de problemas de regressão, tem-se a previsão do tempo, que utiliza dados atmosféricos contínuos para tentar prever a possibilidade do clima. Para problemas de classificação, existe o problema de identificação de animais, apresentado anteriormente, e o tema de classificação de doenças, que, baseado em sintomas, o modelo tenta prever a presença de uma enfermidade (HURWITZ; KIRSCH, 2018a).

Logo, o objetivo principal do aprendizado supervisionado é encontrar padrões que liguem a entrada e a saída do modelo de AM. Dentre os vários métodos utilizados no aprendizado supervisionado estão incluídos: classificadores Naive Bayes (WEBB; KEOGH; MIIKKULAINEN, 2010), regressão linear (YAN, 2009), regressão logística (TOLLES; MEURER, 2016), florestas aleatórias (HO, 1995), máquinas de vetores de suporte (CORTES; VAPNIK, 1995) e redes neurais (RUSSELL; NORVIG; CANNY, 2003). O método de redes neurais será tratado na Seção 2.3, pois ele será importante para o assunto de neuro-evolução tratado nesse trabalho.

2.2.2 APRENDIZADO NÃO-SUPERVISIONADO

Aprendizado Não-Supervisionado, também conhecido como Aprendizado de Máquina Não-Supervisionado, é definido por técnicas que tentam encontrar estruturas escondidas e proximidades entre dados (MARSLAND, 2011). Essa abordagem trata conjuntos de dados que, ao contrário do aprendizado supervisionado, não possuem uma variável específica a ser descoberta (e.g. filmes recomendados) (BRINK JOSEPH RICHARDS, 2016). O propósito dessa abordagem é, com base na similaridade de características, descobrir estruturas ocultas ou grupos de dados sem a necessidade de intervenção externa (e.g. o usuário-professor citado na Seção 2.2.1). Esse tipo de aprendizado é tipicamente utilizado nos algoritmos de recomendação presentes nas mídias sociais de foto e vídeo. Possuindo uma grande quantidade de dados dos usuários, essas plataformas agrupam as pessoas utilizando os históricos de navegação e informações dos usuários (como gostos, tendências etc.). Uma vez os grupos criados, é possível recomendar conteúdos que outras pessoas do mesmo perfil do indivíduo possam preferir (HURWITZ; KIRSCH, 2018a). Além disso, o aprendizado não-supervisionado é a solução ideal para análise de dados, como, por exemplo, estratégias de venda cruzada, reconhecimento de imagem etc. (IBMCLOUD, 2020b).

Uma das principais aplicações do Aprendizado Não-Supervisionado é a busca por padrões entre indivíduos para formar grupos. Isso facilita a compreensão de grandes quantidades de dados. Existem duas classes principais de aprendizado não-supervisionado, o *Clustering* e o *Dimensionality reduction* (BRINK JOSEPH RICHARDS, 2016):

- *Clustering* envolve usar os dados para descobrir grupos e, por conseguinte, dividi-los. Como exemplo de algoritmos tem-se o *k-means* (ABSTRACTS, 1965), modelos de mistura gaussiana (*Gaussian mixture models*) (PRESS SAUL A. TEUKOLSKY, 2007) e agrupamento hierárquico (*hierarchical clustering*) (MAIMON, 2005).

- *Dimensionality reduction* transforma os dados e os reduz em coordenadas-características que capturam a variabilidade dos dados. Algoritmos dessa abordagem abrangem: *Multidimensional Scaling* (MDS) (BORG, 2005), *Principal Component Analysis* (PCA) (BRO; SMILDE, 2014) e o *Manifold Learning* (IZENMAN, 2012).

2.2.3 APRENDIZADO POR REFORÇO

Aprendizado por Reforço, também conhecido como Aprendizado de Máquina por Reforço, é um modelo de aprendizado de comportamentos (OTTERLO; WIERING, 2012). Esse tipo de abordagem, comparado às duas outras aprendizagens vistas nas Seções 2.2.1 e 2.2.2, não demanda conjuntos de dados para treinos. Ao invés disso, esse modelo adota uma abordagem de tentativa e erro. A procura de uma solução é uma parte fundamental desse tipo de aprendizado. O algoritmo busca testar, em um espaço de possibilidades, todas as combinações possíveis de entradas e saídas para maximizar uma recompensa (HURWITZ; KIRSCH, 2018a).

Essencialmente, o aprendizado por reforço traça que existem **estados** (ou situações) em que ocorrem **ações** por um **agente** dentro de um **meio**, esse indivíduo tenta agir para tentar maximizar uma **recompensa** e, ao consegui-la, ele passa a ter um **policiamento** de como agir (MARSLAND, 2011). Então:

- Meio: São as condições às quais o agente está submetido.
- Estado: É uma representação do meio em que o agente está. Não necessariamente o indivíduo possui informação completa sobre o meio. Por exemplo, uma pessoa que apenas possui a visão não conseguirá confirmar se há uma pessoa atrás da porta.
- Agente: É o indivíduo principal executando as ações.
- Ação: O agente, reagindo aos estados, executa ações para atingir um objetivo, recebendo recompensa em contrapartida.
- Recompensa: Com a proximidade do objetivo, o agente recebe recompensas para indicar que aquelas foram ações boas. É válido notar que, dependendo da situação, a recompensa pode vir com atraso. Por exemplo, um indivíduo em um labirinto não saberá se acertou o caminho até que saia dele (MARSLAND, 2011).
- Política: Quando um indivíduo recebe uma recompensa por um estado, é provável que execute essa mesma ação quando o estado ocorrer novamente.

Um exemplo comum para o aprendizado por reforço é o do robô (MARSLAND, 2011): as leituras de sensores de um robô (**agente**) definem o **estado** em que ele está, e, por consequência, o seu **meio**. As possibilidades de movimentação dos motores determinam as suas **ações** e a capacidade de andar sem colidir com o meio é a sua **recompensa**. Algoritmos como Q-learning (WATKINS; DAYAN, 1992), SARSA (RUMMERY; NIRANJAN, 1994) e Monte Carlo (MORAL; MICLO, 2000) são exemplos de algoritmos de aprendizado por reforço.

2.3 REDES NEURAIS ARTIFICIAIS

Redes neurais (RN), ou também chamadas de Redes Neurais Artificiais (RNA), é uma subseção de Aprendizado de Máquina e é inspirada por neurônios biológicos que os animais possuem (RUSSELL; NORVIG; CANNY, 2003). Ela é composta por grupos de nós e conexões, que serão explicados nesta seção. A principal funcionalidade de uma RN é ser treinada com amostras, que possuem suas respectivas características e resultados, para que possa aprender como esse objeto alcança seu resultado, com a finalidade de gerar predições precisas desse objeto. Para isso, a técnica de RNA utiliza-se do modelo de Neurônio Perceptron de McCulloch-Pitts (MCCULLOCH; PITTS, 1943).

No neurônio de McCulloch-Pitts existem quatro conceitos a serem destacados que serão importantes para o conceito de rede neural: conexões de entrada, função de entrada, função de ativação e saída.

- Conexões de entrada: Esse componente compreende as entradas do neurônio, $x_1, x_2, ..., x_n$. Em um neurônio, essas entradas são importantes para fazer ligação com a função de entrada. Para dar importância e diferençar cada uma delas, definiu-se que cada entrada pode ter um peso n que deve ser multiplicado por ela.

- Função de entrada: Ela é definida como uma equação que fará associação entre todas as conexões de entrada. Geralmente, utiliza-se apenas o somatório. Entretanto, essa função pode ser modificada. A função pode ser definida como: $\sum_{i=1}^{n} x_i n_i = x_1 n_1 + x_2 n_2 + ... + x_n n_n$

- Função de ativação: Conhecida por conter a lógica de limiar (*threshold logic*), é a parte mais importante de um neurônio. Depois de realizar a equação da função de entrada, o resultado é inserido na função de ativação. Existem várias funções possíveis a serem utilizadas, como, por exemplo, a identidade, degrau, ReLU (NAIR; HINTON, 2010) etc. No entanto, todas essas funções possuem uma funcionalidade em comum: quando uma combinação de linear de entradas excede um "limiar", ela "dispara", ou seja, a saída será LIGADO (1, um), ao invés de DESLIGADO (0, zero) (RUSSELL; NORVIG; CANNY, 2003).

- Conexões de saída: Por fim, o resultado obtido pela função de ativação será distribuída a qualquer neurônio que estiver conectado, que pode ser nenhum ou até milhares.

A Figura 2.1 ilustra esse modelo de neurônio. Nessa figura, x_n indica as conexões de entradas e os pesos desse neurônio. A função de entrada g agregará as conexões. A função de ativação f fará a decisão. A conexão y é a saída desse neurônio.

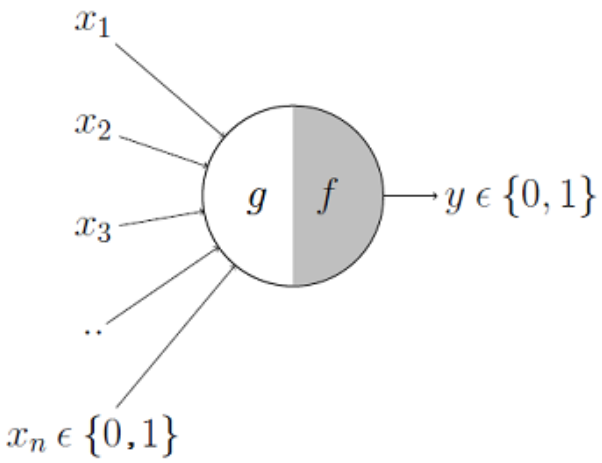

Figura 2.1: Modelo do Neurônio Perceptron de McCulloch-Pitts (CHANDRA, 2018).

De modo simples, uma rede neural é uma coleção de unidades de neurônios conectadas. As propriedades dessa rede são determinadas pela topologia e pelas características de cada neurônio (RUSSELL; NORVIG; CANNY, 2003). Na rede neural, existem nove conceitos importantes que são listados a seguir e ilustrados na Figura 2.2.

- Nós: Dizem respeito aos neurônios (Figura 2.1) de uma rede neural. Na Figura 2.2, eles são representados por elipses e quadrados, indicados pelos números de 1 a 6. Com exceção dos nós de entrada 1 e 2, os nós são equivalentes ao neurônio apresentado na Figura 2.1.

- Conexões: São as ligações existentes entre os nós, cada um deles possui um peso. Na Figura 2.2, eles representados por $w_{i,j} \in \{(i,j) \in \mathbb{N}\}$.

- Entradas: Referem-se aos nós presentes no início da rede neural. Eles são diferentes de nós normais, pois são eles que recebem estímulos e interagem com os dados externos, assim, esses nós não necessitam das "conexões de entrada". Eles podem variar em quantidade de acordo com a necessidade do problema. Na rede neural da Figura 2.2, são os neurônios 1 e 2.

- Saídas: Referem-se aos nós presentes no fim da rede neural. São eles que geram os resultados da rede neural, a partir das entradas. Assim como os nós de entrada, eles também podem variar em quantidade de acordo com a necessidade. Na rede neural da Figura 2.2, são os neurônios 5 e 6.

- Camada: Entende-se que, em uma rede neural, são conjuntos de nós arranjados em "camadas" e formam a topologia da rede. Na Figura 2.2, camadas são os conjuntos: 1,2; 3,4 e 5,6.

- Genoma: Refere-se às características presentes em uma rede neural — a quantidade de neurônios, o seu arranjo, as conexões existentes, os pesos, as funções de entrada e a função de ativação.

- Genótipo: São as informações presentes no genoma da rede neural.

- Topologia: É o formato, estrutura, ou arranjo de neurônios que uma rede neural apresenta. Por exemplo, na Figura 2.2, descreve-se que os nós 1 e 2 são as entradas, 5 e 6 são as saídas e 3 e 4 são uma camada interna. Além disso, a saída do nó 1 está conectada ao nó 3 pela conexão de peso $w_{1,3}$, a saída do nó 2 está conectada ao nó 3 pela conexão de peso $w_{2,3}$ e assim por diante.

- Fenótipo: é uma a rede neural resultante da junção da topologia e do genótipo de uma rede.

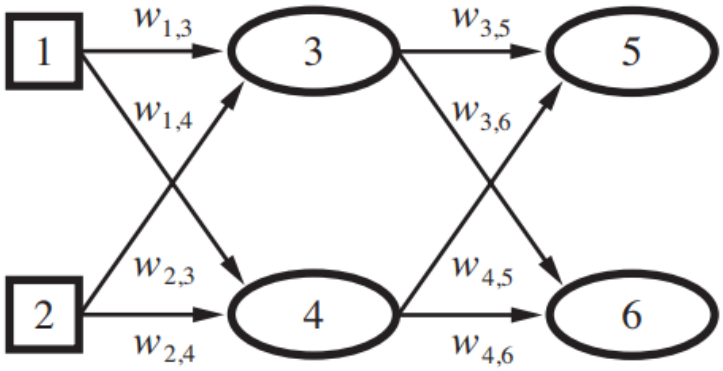

Figura 2.2: Imagem de uma rede neural (RUSSELL; NORVIG; CANNY, 2003).

Na Figura 2.2, os números 1 a 6 representam os nós. As conexões de cada nó são representadas por $w_{i,j}$. Os nós 1 e 2 são chamados de nós de entrada. Os nós 5 e 6 são os nós de saída. Os nós 3 e 4 são os nós ocultos. Todos esses nós estão distribuídos em camadas, entrada, saída e oculto. Genomas são as características presentes na rede neural. Genótipo são as informações presentes no genoma. Topologia é o arranjo de neurônios. Fenótipo é a junção da topologia e do genótipo.

Tipicamente, os neurônios de uma rede neural são organizadas em camadas e cada camada pode possuir ou não um genótipo único. Além disso, a quantidade de camadas e o número de nós por camada não são fixos, ou seja, são completamente dependentes de cada situação. Por fim, no funcionamento de uma rede neural, o sinal da primeira camada (camada com nós de entrada) percorrerá as camadas internas até a camada final (camada com nós de saída). Dependendo da topologia e do genótipo, os resultados desse sinal podem variar bastante. A Figura 2.3 ilustra um exemplo de uma rede neural. Existem os nós de entrada, saída e os de camada interna, os três estão ilustrados pelas cores laranja, verde e azul, respectivamente. Além disso, as conexões possuem pesos que levarão um sinal modificado até os nós. O resultado é um fenótipo, rede neural resultante.

Na Figura 2.3, os *Node Genes*, que são as caixas de cima, representam os nós da rede neural e suas camadas. *Connection Genes*, que são as caixas de baixo, determinam os nós que eles

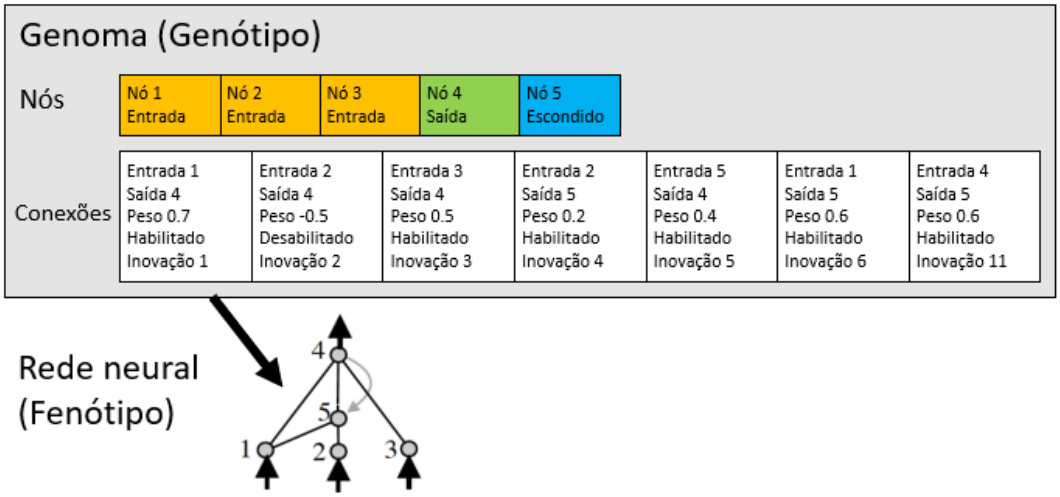

Figura 2.3: Imagem de uma rede neural resultante do genótipo e da topologia (STANLEY; MIIKKULAINEN, 2002).

conectarão e seus pesos.

Além dos nós de entrada, saída e ocultos, algumas redes neurais podem também apresentar o nó enviesado (*bias node*, em inglês). Esse nó, juntamente com os nós de entrada, permitem a mudança da função de ativação através da adição de uma constante. Sendo independente das entradas, esse valor pode transpor o resultado utilizando seu valor constante (MARSLAND, 2011). Um exemplo de nó enviesado pode ser visto na Figura 2.4. O *bias* está sinalizado por um indicador vermelho.

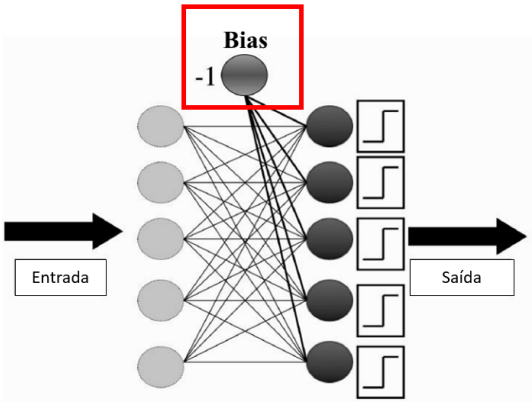

Figura 2.4: Exemplo de uma rede neural perceptron, com um nó *bias* (MARSLAND, 2011).

Redes neurais é um assunto de grande importância nesse trabalho, pois sua combinação com algoritmos genéticos gerará o conceito de neuro-evolução, tratado na Seção 2.4.

2.4 COMPUTAÇÃO EVOLUCIONÁRIA

Computação evolucionária (CE) é um campo que estuda o desenvolvimento de algoritmos inspirados na seleção natural. A grande vantagem do paradigma evolucionário é a criação de algoritmos evolucionários. Para isso, esses algoritmos são projetados utilizando conceitos inspirados na Biologia, como, por exemplo, população, seleção natural, funções de aptidão (*fitness*, em inglês) e evolução dos indivíduos mais aptos (EIBEN; SMITH et al., 2003). Existem quatro sub-áreas principais no campo da computação evolucionária: programação evolucionária, estratégias de evolução, algoritmos genéticos e programação genética (EIBEN, 2015). Na Seção 2.4.1 serão explicadas as funcionalidades, conceitos e objetivos de algoritmos genéticos.

2.4.1 ALGORITMOS GENÉTICOS

Como mencionado na Seção 2.4, Algoritmos Genéticos (AG) são uma sub-área de Computação Evolucionária. Seu objetivo principal é a criação, reprodução e mutação de indivíduos a fim de produzir soluções para problemas (EIBEN, 2008). O código mostrado no Algoritmo 1 apresenta uma pseudo-codificação, que expressa os passos utilizados nos algoritmos genéticos.

Algoritmo 1 Pseudo-código de um algoritmo evolucionário, retirado de (EIBEN, 2015)

```
INICIALIZAR população de candidatos com soluções aleatórias;
AVALIAR cada candidato;
enquanto CONDIÇÃO DE TÉRMINO não é satisfeita faça
    SELECIONAR candidatos;
    RECOMBINAR genética com os candidatos-pais;
    MUTAÇÃO da prole resultante;
    AVALIAR os novos canditatos;
    SELECIONAR indivíduos para a proxima geração;
fim
```

A partir desse código, pode-se notar o método de funcionamento dos AG, que é manipular uma população de indivíduos com genótipos diferentes para gerar resultados diferentes, assim, tentando encontrar um meio de solucionar o problema. Existem seis conceitos principais que podem ser retirados desse algoritmo: representação cromossômica, iniciação, avaliação de aptidão, seleção, cruzamento e mutação (EIBEN, 2015).

- Representação cromossômica: Também conhecido como característica, solução do indivíduo ou genética. Descreve as soluções possíveis que podem ser codificadas no indivíduo da população.

- Iniciação: No começo de um AG, a população de soluções possíveis é gerada. É válido notar que o número de indivíduos dentro da população inicial não é definido por um número fixo, ele pode ser escolhido de acordo com a necessidade do problema. Além disso, as soluções possíveis dos indivíduos, ou seja, sua representação ou suas características, são iniciadas aleatoriamente.

- Avaliação de aptidão (*fitness*): Esse conceito baseia-se em fazer comparações e análises das soluções propostas pelos indivíduos. Para isso, é usado algo chamado função de aptidão (*fitness function*, em inglês). Essa função calcula o quão próximo uma solução está do objetivo estabelecido pelo problema. Ou seja, em um jogo simulador de corrida, se os indivíduos fossem os carros e as soluções fossem a sequência de ações que o carro desempenha (e.g. acelerar, virar para a direita, etc.), a avaliação de aptidão seria o quão rápido e o quão perto o carro está da linha de chegada. Então, para o exemplo do jogo de corrida, a função de aptidão seria calculada a partir das informações do veículo e da corrida: posição do carro no placar de colocados, sua velocidade desempenhada, quantidade de colisões, tempo decorrido e a distância do carro até a linha de chegada. É válido mencionar que alguns dos elementos apresentados para a função de aptidão são proporcionais e outras são inversamente proporcionais. Dentro de uma função, é possível ter variáveis que estimulem e outros que desestimulem o comportamento de um indivíduo, afetando sua aptidão.

- Seleção: É a noção de escolher os indivíduos da população mais aptos a solucionar o problema. Normalmente, as soluções apresentadas por esses indivíduos possuem a melhor pontuação de aptidão, derivada da função de aptidão. Desse modo, maiores são as chances que esse indivíduo tem de passar suas "características" para a prole.

- Cruzamento: Esse é o processo de combinar as representações de duas soluções diferentes

para gerar uma nova prole. A ideia por trás desse mecanismo é que combinar diferentes organismos pode resultar em filhos que compartilham de características dos pais, portanto, passando a solução para frente e melhorando a aptidão.

- Mutação: Por último, mutação tem como objetivo aumentar a diversidade de uma população. Assim como na Biologia, que a mutação tem a funcionalidade de gerar características novas onde não havia antes, a mutação em AG tem por objetivo introduzir novos elementos que não existiam previamente no sistema.

É importante destacar que os AG possuem alguns parâmetros importantes. O tamanho da população inicial e o tamanho da população reproduzida determinam o número de indivíduos dentro das respectivas ocasiões. A taxa de mutação indica a chance em porcentagem de uma mutação ocorrer. Essa taxa é muito importante pelo fato de inserir mudanças ao meio, porém ele deve ser manipulado com cautela, um valor baixo afetará a diversidade da população, enquanto que um valor alto atrapalhará a finalização da solução. Por fim, a taxa de cruzamento determina a quantidade de cruzamentos dos parentes (HOLLAND, 2012).

Finalmente, após apresentar os conceitos relevantes a algoritmos genéticos e rede neural, é possível debater o cerne desse trabalho monográfico: Neuro-evolução, tratado na Seção 2.5.

2.5 NEURO-EVOLUÇÃO

Baseado na evolução biológica de sistemas nervosos, Neuro-evolução (NE) é uma forma de Inteligência Artificial que utiliza técnicas de Computação Evolucionária para criar redes neurais artificiais. Além de ter uma ampla aplicabilidade, neuro-evolução é muito utilizada, pois muitos dos problemas relacionados à IA podem ser considerados como problemas de otimização, que é uma das especialidades da CE e, por consequência, da NE. O fato dessa técnica ser inspirada na teoria evolutiva e nas metáforas biológicas fornece a ela uma base sólida (RISI; TOGELIUS, 2017).

Treinar redes neurais artificiais envolve a procura de parâmetros adequados para a topologia, conexões e pesos das redes neurais. A ideia básica da NE é treinar uma rede neural utilizando

recursos de algoritmos genéticos. Em outras palavras, modificar a genética e topologia do individuo (rede neural) para encontrar o melhor desempenho possível para uma dada tarefa. Para isso, o conceito de *fitness*, ou também chamado de aptidão, foi trazido da Computação Evolucionária. Assim como na CE, *fitness* em Neuro-evolução é um número usado para fazer comparação entre indivíduos, assim determinando que indivíduo obteve mais sucesso em fazer determinada tarefa (LEHMAN; MIIKKULAINEN, 2013). Para calcular o valor da aptidão, é necessário medir o quão próximo do objetivo final a rede neural alcançou. Diferentes algoritmos e diferentes situações empregam métodos de cálculo de aptidão diferentes. Por exemplo, em jogos, dependendo do objetivo do criador, a movimentação mais eficiente possível pode não ser a desejada e, portanto, a abordagem da aptidão pode mudar completamente.

Comparado a outros métodos de aprendizado de redes neurais, Neuro-evolução é uma técnica pouco estrita. Em seu treinamento, ela permite aprendizado sem objetivos explícitos, com *feedback* reduzido e com o uso de topologias arbitrárias de redes neurais (LEHMAN; MIIKKULAINEN, 2013). Tomando como exemplo o caso do jogo de carro, ao contrário dos algoritmos genéticos, que necessitam das informações do carro, pista, ocorrência de colisão etc., a neuro-evolução trabalha apenas com as ações possíveis que o carro consegue desempenhar.

A técnica de neuro-evolução é bastante utilizada para a resolução de problemas de aprendizado por reforço e é comumente aplicada em treinamento de robôs e em IA de jogos (FLOREANO; DURR; MATTIUSSI, 2008).

2.5.1 NEURO-EVOLUÇÃO EM JOGOS

Em jogos, o uso de neuro-evolução para geração de redes neurais já se mostrou bastante útil para a automação e adaptabilidade de Inteligências Artificiais, tanto em jogos físicos (e.g. xadrez e jogos de cartas) quanto em jogos digitais (e.g. jogos de fliperama, jogos de estratégia, jogos de plataforma) (HAUSKNECHT MATTHEW; LEHMAN, 2014; PARKER; BRYANT, 2008). A presença da evolução neuro-genética é importante, pois consegue trazer uma variedade de ações que conseguem aperfeiçoar a experiência de quem está jogando. Elementos, como, por exemplo, seleção de ações, avaliação de estados, construção de estratégias, gerador de conteúdo

e customizações para o jogador (RISI; TOGELIUS, 2017).

- Seleção de ações: A NE é capaz de controlar adequadamente as ações do jogador baseando-se na grande quantidade de informações que chegam até ele. Jogos como *Quake II* e *Flappy Bird* já tiveram redes neurais feitas com neuro-evolução para jogá-los (PARKER; BRYANT, 2008; ZHU, 2020; KI; LYU; OH, 2006).

- Avaliação de estados: Um dos primeiros usos encontrados para a NE foi o de estimar o valor das peças e posições estratégicas para jogos clássicos de tabuleiro. Em jogos como *Xadrez*, *Damas* e *Go*, a rede neural valora numericamente os movimentos. Esse valor está diretamente relacionado a quão próximo o jogador está da vitória. (STANLEY; MIIKKULAINEN, 2004; FOGEL et al., 2004)

- Construção de estratégias: Devido à natureza de topologias arbitrárias, a NE consegue formar vários tipos de solução para o mesmo problema, o que ele chama de nichar (*niching*, em inglês). Isso permite que existam várias formas de jogar e diferentes conjuntos de estratégias. Jogos de mesa como *Texas Hold'em Poker* e de computador como *EvoCommander* são exemplos desse tipo de nicho (LOCKETT; MIIKKULAINEN, 2008; JALLOV; RISI; TOGELIUS, 2016).

- Gerador de conteúdo: O uso de IA nesse tipo de ambiente permite a criação procedural de conteúdo para jogos. Jogos como *Petalz* (RISI et al., 2016), que gera variedades de plantas, e *Creatures* (GRAND et al., 1997), que simula animais virtuais e suas ações, certamente seriam muito mais difíceis de serem realizados se fossem feitos pelos métodos tradicionais de aprendizado.

- Customizações para o jogador: A neuro-evolução e aprendizado do usuário já foram utilizados para tentar capturar os comportamentos do jogador. Um experimento utilizando o jogo *Super Mario Bros.* foi feito para descobrir se uma rede neural treinada conseguiria predizer qual nível do jogo o jogador iria querer (PEDERSEN C.; TOGELIUS, 2010). Esse experimento reuniu dados dos jogadores, níveis, tempo de jogo e preferências de jogo

do jogador e conseguiu predizer com mais de 90% de acerto a preferência dele. Assim, é possível criar uma outra versão do mesmo jogo que tenha apenas as fases de preferência do usuário.

2.6 CONCLUSÃO

Nesse capítulo, foram apresentados diversos temas que fornecem base a essa monografia. Assuntos como aprendizado de máquina, redes neurais e algoritmos genéticos são muito importantes para a definição do que é Neuro-evolução. Além disso, o uso da NE em jogos mostra como ela pode ser útil para os objetivos desse trabalho.

Depois de discutir sobre a Neuro-evolução e suas aplicações possíveis em jogos, o Capítulo 3 discorrerá sobre trabalhos que utilizam a neuro-evolução em suas aplicações.

Capítulo 3

TRABALHOS RELACIONADOS

Na Literatura, não é incomum encontrar aplicações de neuro-evolução em jogos. Enquanto alguns autores desenvolvem abordagens e definem diferentes estratégias para a maneira de jogar o jogo (LOCKETT; MIIKKULAINEN, 2008), outros criam redes neurais para conseguir jogá-los autonomamente (PARKER; BRYANT, 2008). As seções seguintes apresentam alguns exemplos da aplicação de neuro-evolução.

3.1 NERO

O trabalho "*Real-Time Neuroevolution in the NERO Video Game*" (STANLEY; BRYANT; MIIKKULAINEN, 2005) é um exemplo porque apresenta uma implementação de neuro-evolução em que há uma evolução da Inteligência Artificial de exércitos aliados e inimigos. O método aplicado ao jogo procura desenvolver a complexidade de um jogo através do tempo.

No trabalho, há o desenvolvimento do jogo chamado NERO, sigla para *Neuro-Evolving Robotic Operatives*. Nesse jogo, o jogador assume o papel de um treinador, que deve ensinar habilidades para robôs controlados por Inteligência Artificial. No desenrolar do jogo, o exército aprende, através de neuro-evolução, a melhorar sua movimentação e a encontrar alternativas para dificuldades encontradas. No início do jogo, o exército aliado não possui habilidades, apenas a capacidade de aprender através da neuro-evolução. Para preparar uma evolução da Inteligência Artificial, o jogo permite que o jogador crie diferentes níveis para desafiar seu

exército aliado. Idealmente, o jogador deve criar níveis fáceis e aumentar paulatinamente sua dificuldade para que o exército consiga desenvolver habilidades úteis. Ao final do aprendizado, o exército aliado é posto contra um outro exército inimigo de outro jogador para que se possa verificar quem tem o melhor exército. Um exemplo desse desenvolvimento e criação de níveis é mostrado na Figura 3.1.

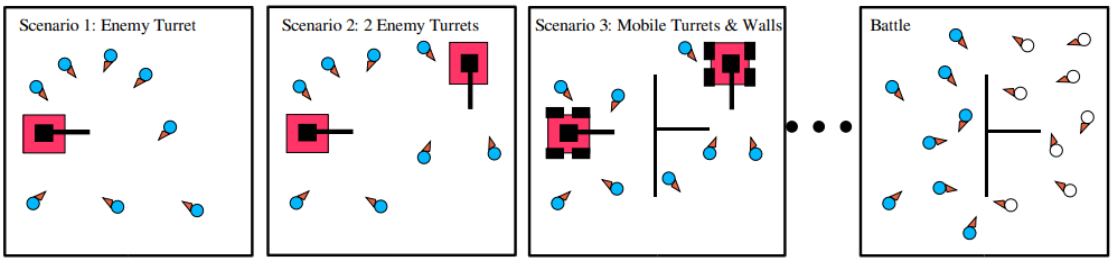

Figura 3.1: Esquema de aumento de dificuldade de níveis dentro do jogo NERO, dividido em cenários (STANLEY; BRYANT; MIIKKULAINEN, 2005).

O jogo foca principalmente na criação de exércitos para a estimular a competitividade entre jogadores. No entanto, o destaque e a mecânica principal do trabalho é o uso de neuro-evolução para criar comportamentos dos exércitos e cumprir certos objetivos, como, por exemplo, derrotar o outro jogador.

3.2 *SELF-LEARNING CHESS*

O trabalho "*A Self-Learning Evolutionary Chess Program*" (FOGEL et al., 2004) apresenta esquemas e técnicas de neuro-evolução para o desenvolvimento de redes neurais que possam jogar xadrez em alto nível.

Para a execução do trabalho, os autores desenvolveram uma rede neural que aprende, através da evolução, a otimizar seus movimentos e a fazer jogadas mais inteligentes. No início, a Inteligência Artificial sabe pouco e comete muitos erros, porém, quanto mais a rede neural joga, mais ela melhora suas estratégias de jogo. Para fazer isso de maneira rápida, os responsáveis pelo trabalho decidiram criar múltiplos jogadores-máquina que seriam postos para jogar entre si. Com isso, as máquinas continuaram melhorando em uma velocidade acelerada e eventualmente estariam jogando em nível de especialistas.

Ao fim do experimento, os autores colocaram a melhor solução gerada para jogar contra o algoritmo do jogo *Chessmaster 8000* (ABANDONWARE, 2000), um jogo de xadrez conhecido por desempenhar em alto nível. Em média, essa solução apresentou resultados que alcançavam 2437 na escala da Federação de Xadrez dos Estados Unidos, o que representa um Mestre Sênior. Essa pontuação foi de 400 pontos maior que programas semelhantes que não passaram pelo processo de neuro-evolução.

O principal destaque deste trabalho é a demonstração de como a neuro-evolução consegue ser flexível em seu aprendizado, gerar diferentes estratégias e criar redes neurais que se desenvolvem de acordo com a exigência e situação de oponentes mais habilidosos.

3.3 ATARI

O artigo "*A Neuroevolution Approach to General Atari Game Playing*" (HAUSKNECHT MATTHEW; LEHMAN, 2014) aborda o desafio de uma neuro-evolução aprender a jogar o *Atari 2600* (ABANDONWARE, 2000). O console foi escolhido por possuir uma variedade de jogos, desde Damas, *Pitfall*, *Space Invaders* até *Pac-man*.

Diferente dos outros, esse trabalho focou mais na comparação de quatro algoritmos diferentes, *Conventional Neuro-Evolution*, *CMA-ES*, *NEAT* e *HyperNEAT* (CLUNE et al., 2011; HANSEN; OSTERMEIER, 2001; STANLEY; MIIKKULAINEN, 2002; RISI; STANLEY, 2012). Além disso, os autores desenvolveram redes neurais utilizando esses algoritmos, que conseguiram evoluir eficientemente e criaram diferentes maneiras de terminar o mesmo jogo.

A Figura 3.2 ilustra as pontuações obtidas dentro do jogo e ela traz uma comparação das abordagens utilizadas no artigo. Da esquerda para direita na Figura 3.2, estão comparados a otimização aleatória (*Random*), HyperNeat, NEAT, *Planners* (redes que possuem acesso à estados futuros do jogo) e jogadores especialistas humanos (possuem familiaridade e experiência no jogo). Na figura, é demonstrado que o NEAT possuía uma média de pontuação levemente superior ao ser comparado com os outros algoritmos, como pode ser observado no eixo Z-score da figura. Contudo, pode-se observar que as pontuações feitas por humanos especialistas,

em média, foram quase sempre maiores. Isso comprova que, apesar dos algoritmos de neuro-evolução gerarem soluções que podem não ser melhores que as feitas por humanos, eles são capazes de encontrar saídas e oportunidades que despenham melhor que a média humana.

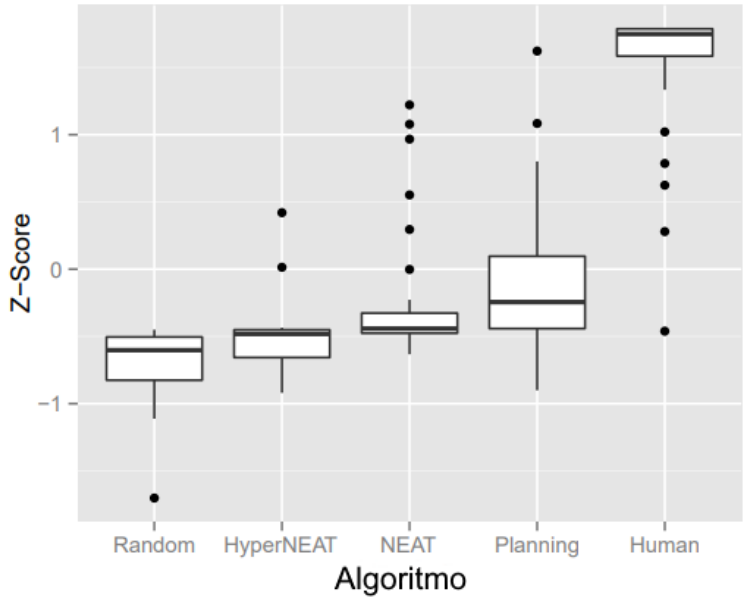

Figura 3.2: Comparação de algoritmos de neuro-evolução (HAUSKNECHT MATTHEW; LEHMAN, 2014).

A conclusão do artigo é que a neuro-evolução se mostra uma abordagem promissora ao *General Video Game Playing* (GVGP), um tema que tem como objetivo projetar um agente de IA que seja capaz de jogar múltiplos jogos eletrônicos sem intervenção humana (PEREZ-LIEBANA et al., 2018). Isso indica que a neuro-evolução pode gerar soluções para diversos problemas diferentes, por vezes desempenhando melhor que humanos.

3.4 *NEUROEVOLUTION IN GAMES*

O artigo "*Neuroevolution in Games: State of the Art and Open Challenges*" (RISI; TOGELIUS, 2015) disseca como a neuro-evolução pode ser aplicada em elementos para agregar ao jogo, como Inteligência Artificial dos oponentes ou gerador de conteúdo.

O escopo principal desse artigo é fazer uma compilação de artigos científicos que utilizam a neuro-evolução dentro dos jogos. No processo, os autores analisaram essas aplicações e os

separaram em cinco áreas diferentes: avaliação de estados, seleção de ações, seleção de diferentes estratégias, gerador de conteúdo e modelagem de experiência do jogador. Em seguida, eles descreveram os tipos de redes neurais que são utilizados nos artigos (e.g. perceptron multi-camada e redes recorrentes), os tipos de algoritmos de neuro-evolução nas abordagens direta (e.g. CMA-ES (HANSEN; OSTERMEIER, 2001)) e indireta (e.g. CPPNs (STANLEY, 2007)), os métodos utilizados pelos artigos para avaliar o *fitness* das redes (e.g. evolução incremental (GOMEZ; MIIKKULAINEN, 1997)) e os procedimentos empregados para definir as entradas da rede (e.g. sensores e controles do jogo). Por fim, o artigo cita importantes desafios que esse tema enfrenta como, por exemplo, a otimização para alto desempenho, uso de dados de dimensões maiores e métodos eficientes para usos comerciais em jogos.

A importância desse trabalho é que ele reúne vários artigos e mostra como cada um deles utiliza a neuro-evolução para melhorar aspectos nos jogos. Isso mostra como a neuro-evolução pode ser empregada na conclusão automática de jogos e na melhoria da experiência do jogador. No entanto, apesar de utilizar as saídas das redes neurais para melhorar a experiência do jogador (por exemplo, gerando o número de inimigos em um determinado nível), o trabalho não aborda como essas informações (número de inimigos) podem ser utilizadas de maneira benéfica, como, por exemplo, ao avaliar o nível de dificuldade de um jogo.

3.5 CONCLUSÃO

Apesar de haver muitas aplicações de neuro-evolução em jogos, seu uso para conseguir resultados secundários como o "número de passos dados" ou "número de botões pressionados" não é bem desenvolvido. O trabalho apresentado na Seção 3.4 faz uma compilação de utilizações de Neuro-evolução, porém, ele não aborda como a própria rede neural gerada ou as informações de execução da mesma, como, por exemplo, informações de controle ou tempo de tela, podem ser utilizados de forma benéfica.

O capítulo 4 entrará em detalhes sobre os tipos de algoritmos existentes que podem ser usados.

Capítulo 4

ALGORITMOS PARA NEURO-EVOLUÇÃO

Na área de neuro-evolução, existem diversos métodos que são amplamente testados e utilizados. Esses algoritmos podem ser divididos em dois grupos: os algoritmos que apenas evoluem os pesos das conexões de uma rede de neuro-evolução tradicional, as chamadas redes de topologia fixa; e aqueles que evoluem ambos, topologia e pesos, chamados em inglês de *TWEANNs (Topology and Weight Evolving Artificial Neural Network)* (STANLEY; MIIKKULAINEN, 2002). As próximas seções detalham os TWEANNs, suas características, subdivisões e alguns exemplos de cada grupo.

4.1 ALGORITMOS DE NEURO-EVOLUÇÃO

Os TWEANNs podem ser divididos em dois grupos principais, aqueles que utilizam codificação direta e os que usam codificação indireta. Os que utilizam o esquema de codificação direta compõem o maior grupo. Neles são especificadas os pesos de cada conexão e cada nó que vai aparecer na rede neural. Como exemplo tem-se a Neuro-evolução genética (RONALD; SCHOENAUER, 1994; ANGELINE et al., 1994; STANLEY; MIIKKULAINEN, 2002). Além disso, pode-se dividir o grupo de codificação direta em três partes: codificação binária, codificação por grafo e codificação não cruzada. Esses grupos serão mais detalhados junto com um exemplo

de algoritmo nas próximas seções.

Em contraste com a codificação direta, na codificação indireta os genomas estipulam regras de como construir a topologia. Utilizando um conjunto de instruções, a rede deve ser capaz de dividir, adicionar, copiar e remover conexões e nós do sistema (GRUAU, 1994), modificando indiretamente aquela rede específica. A codificação indireta é mais compacta que a direta porque as conexões e nós não são especificados no genoma, mas derivados. Como exemplo de algoritmos indiretos temos o *Cellular Encoding* (GRUAU, 1994; MANDISCHER, 1993).

As seções 4.1.1 até 4.1.4 detalham alguns algoritmos de neuro-evolução e seus tipos de codificação.

4.1.1 ALGORITMO GENÉTICO ESTRUTURADO (AGE)

Começando pela abordagem direta, tem-se a codificação binária. É a implementação mais fácil pelo fato de usar representações vetoriais e matriciais para representar os nós e as conexões.

Como exemplo de codificação binária, tem-se o algoritmo genético estruturado (*Structured Genetic Algorithm*, em inglês — sGA) de McGregor, um dos primeiros métodos de neuro-evolução (DASGUPTA; MCGREGOR, 1993). Seu algoritmo é conhecido pela simplicidade e facilidade de uso através de vetores e matrizes, porém, ele é limitado em alguns aspectos, como, por exemplo:

1. Possui um limite de conexões estabelecido pelo quadrado do número de nós, consequência do uso de matrizes e vetores. Isso faz com que esse método seja considerado limitado para os padrões atuais (STANLEY; MIIKKULAINEN, 2002).

2. Por causa da maneira que foi codificado, o número de nós da rede neural possui uma quantidade máxima a ser usada e esse número é compartilhado por todas as redes do treinamento. Além disso, os neurônios e conexões devem ser pré-determinados.

3. A utilização de vetores para representar estruturas de grafos torna o cruzamento difícil e com poucas combinações úteis. (DASGUPTA; MCGREGOR, 1992)

4.1.2 DISTRIBUIÇÃO GENÉTICA PARALELAMENTE DISTRIBUÍDA (DGPD)

Seguido pela abordagem binária, a codificação por grafos é a estrutura que constitui o maior grupo de TWEANNs. Ao contrário dos algoritmos binários que armazenam seus dados em forma de vetores, essa estrutura armazena suas informações em estruturas de grafos. Graças a essa nova abordagem de organização, algoritmos como o DGPD ganharam novas funcionalidades sobre aquelas com codificação binária.

O algoritmo de Distribuição Genética Paralelamente Distribuída (DGPD) (em inglês, *Parallel Distributed Genetic Programming*, PDGP) (PUJOL; POLI, 1998) é um exemplo da codificação por grafos. Os autores utilizaram um sistema de dupla representação, em que a topologia era apresentada como uma estrutura de grafos e um vetor representava as conexões dos nós. Graças a esse tipo de estrutura, foi possível implementar funções como o cruzamento de sub-grafos e mutação de estrutura da topologia na representação em grafos, enquanto havia mistura e mutação de conexões utilizando a apresentação linear dos vetores.

Assim como o AGE (Seção 4.1.1), essa abordagem também contêm problemas relacionados ao número de nós na rede, que é limitado pelo tamanho da matriz representando o grafo. Além disso, o uso de sub-grafos para a reprodução e cruzamento dos indivíduos possui um pequeno defeito: o cruzamento entre diferentes sistemas pode ser imprevisível, trazer perdas de comportamentos existentes e gerar redes não-funcionais (STANLEY; MIIKKULAINEN, 2002).

4.1.3 AQUISIÇÃO GENERALIZADA DE LINKS RECORRENTES (AGNLR)

Pelo fato do cruzamento de diferentes topologias poder trazer perdas de funcionalidades ao sistema, alguns pesquisadores desistiram da ideia de cruzamento por inteiro, dando origem à codificação não-cruzada. Uma característica comum entre os algoritmos que usam essa abordagem é que eles utilizam codificação por grafos em seus algoritmos. Essencialmente, sua única diferença é o abandono da ferramenta de cruzamento.

O algoritmo *Aquisição Generalizada de links recorrentes* AGNRL (*GeNeralized Acquisition of Recurrent Links*, em inglês — *GNARL*) (ANGELINE et al., 1994) é um exemplo de sistema que usa esse paradigma. O algoritmo AGNLR demonstra que o TWEANNs não necessariamente precisam de cruzamento e ferramentas da evolução biológica para funcionar. Contudo, os benefícios demonstrados pela mistura de indivíduos ainda são incrivelmente úteis, atestado pelo fato de que algoritmos modernos não o desprezaram (STANLEY; MIIKKULAINEN, 2002; FRANÇA, 2018).

4.1.4 CODIFICAÇÃO CELULAR (CC)

Em contraste com as codificações diretas, a codificação indireta também é utilizada. Um bom exemplo de algoritmo com paradigma indireto é o Codificação Celular (em inglês, *Cellular Encoding*) (GRUAU, 1994). Ao invés de manipular os pesos de topologias como os métodos diretos fazem, a codificação indireta inspira-se na divisão de células biológicas para criar regras de divisões e desenvolvimento dos nós e conexões.

As vantagens da CC é que esse método consegue ser extremamente compacto comparado com as outras abordagens. No entanto, de acordo com os criadores, codificação indireta requer conhecimento mais profundo em genética e mecanismos neurais (BRAUN; WEISBROD, 1993). Isso significa que, se não houver controle apropriado, o sistema pode gerar vieses abruptos e imprevisíveis que podem passar despercebidos.

4.2 AVALIAÇÃO

Tendo visto os principais tipos de codificação dos TWEANNs, é válido citar que os algoritmos de neuro-evolução mais antigos, incluindo os algoritmos citados nas seções anteriores, demonstram alguns desafios de desenvolvimento. Entre esses desafios está o problema da permutação.

O problema da permutação, ou desafio da competição (RADCLIFFE, 1993), é caracterizado quando há mais de uma rede-neural-solução para o mesmo problema. Durante o desenvolvimento de redes neurais, se duas redes com diferentes estruturas e mesma qualidade de soluções

se cruzam, a descendência dessas soluções tende a ser problemática. Para tentar solucionar esse problema, alguns trabalhos propõem assumir que a prole sairia funcional (PUJOL; POLI, 1998) e outros abandonam a mistura genética por completo (ANGELINE et al., 1994).

Outro desafio que os TWEANNs enfrentam é a proteção de características emergentes de uma população. Quando um novo nó ou conexão é adicionado a uma rede, geralmente, seu desempenho tende a cair porque um novo nó ou conexão dificilmente é criado já otimizado (STANLEY; MIIKKULAINEN, 2002). Por isso, é necessário proteger genes emergentes, pois eles podem ser potencialmente inovadores comparados a outras topologias. O algoritmo AGNLR (Seção 4.1.3) tenta resolver esse problema ao criar nós não funcionais. Ou seja, quando um nó é adicionado nessa rede neural por meio de mutação, ele permanece desconectado da topologia até que a mutação o conecte à estrutura. O problema dessa abordagem é que o próprio autor imaginou que todos os nós em algum momento iriam se conectar à rede, contudo, isso nem sempre acontece.

Por fim, os algoritmos TWEANNs também possuem problema com sua topologia. Na maioria dos sistemas, a população inicial é composta de topologias aleatórias. A princípio, aleatoriedade oferece diversidade às redes neurais, contudo, essa abordagem gera efeitos colaterais imprevisíveis. Por exemplo, é possível que os algoritmos gerem indivíduos que não possuam entrada e saídas conectadas (STANLEY; MIIKKULAINEN, 2002). Esse problema pode ser resolvido pelos próprios algoritmos, porém, um tempo considerável será gasto no processamento.

Algumas ferramentas foram propostas para contornar esses problemas. O algoritmo NEAT oferece algumas soluções dentro da codificação genética que minimizam ou acabam com esses problemas. Isso será abordado na Seção 4.3.

4.3 *NEUROEVOLUTION OF AUGMENTING TOPOLOGIES* (NEAT)

Como apresentado nas seções anteriores, é possível perceber que existem inúmeros algoritmos viáveis para a implementação de neuro-evolução, porém, a maioria deles possui alguma caracte-

rística peculiar, limitação relacionada à topologia (e.g. cruzamento) ou problemas relacionados aos empecilhos mencionados na Seção 4.2. Para tentar solucionar esses problemas, o algoritmo NEAT, do inglês *NeuroEvolution of Augmenting Topologies* (STANLEY; MIIKKULAINEN, 2002), foi criado e suas soluções propostas de otimização de topologias, pesos e conexões são utilizadas até hoje, tanto para algoritmos de evolução quanto para jogos (HEIDENREICH, 2019; PARKER; BRYANT, 2008).

Uma das peculiaridades que o NEAT apresenta é seu esquema de codificação genética. Seu algoritmo permite que, quando dois indivíduos cruzam, os genes, entradas, saídas e conexões correspondentes são preservados. Além disso, graças a essa preservação de genes, a ferramenta de mutação do sistema pode funcionar corretamente. A mutação do NEAT pode adicionar, retirar e modificar os pesos e conexões de estrutura da rede, dessa maneira, cada mutação pode expandir o tamanho da rede e adicionar genomas ao sistema, contribuindo assim para a aptidão da rede neural ao problema (STANLEY; MIIKKULAINEN, 2002). A Figura 4.1 ilustra os dois tipos de mutação estrutural existentes: a que modifica as conexões e a que modifica os nós.

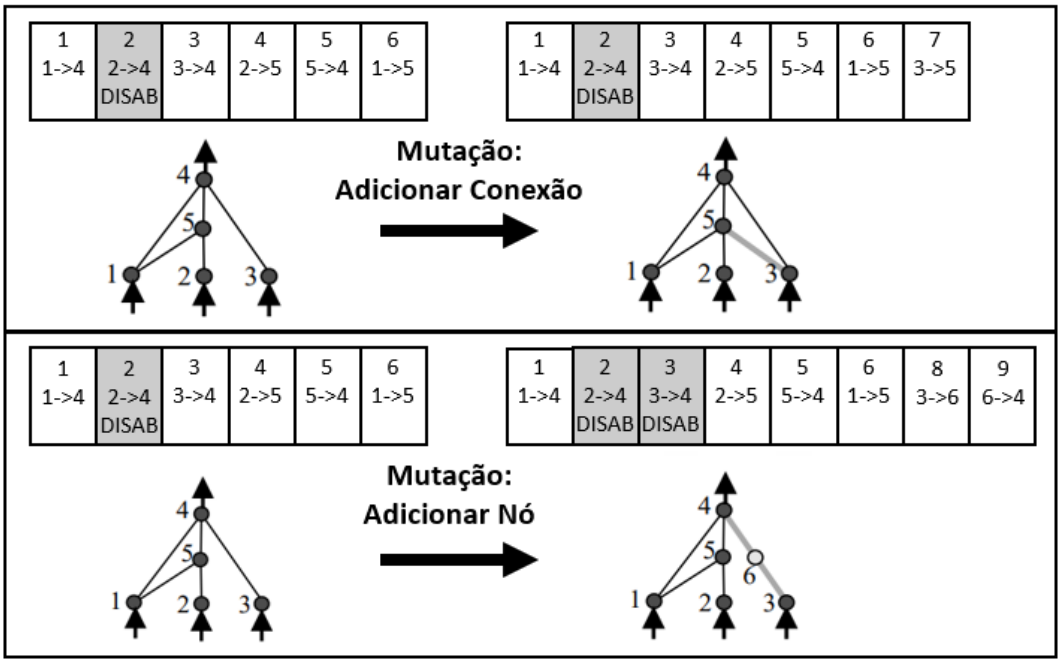

Figura 4.1: Mutação de redes no algortimo NEAT (STANLEY; MIIKKULAINEN, 2002).

Outra característica do NEAT que permite a mutação do sistema é a utilização do identificador de genes, ou número de inovação. Uma informação pouco utilizada na neuro-evolução

dos algoritmos anteriores é a noção exata de quais genes são semelhantes entre indivíduos de uma população. Essa informação é chamada origem do gene. Com a informação da origem, é possível identificar previamente se dois indivíduos possuem genética semelhantes ou não. Com isso, torna-se factível a prevenção do problema de permutação que afeta os TWEANNs e, consequentemente, torna possível o sistema de mutação do NEAT. Esse sistema de identificador pode ser exemplificado na Figura 4.1. Na figura, os genomas, que são as conexões das redes, estão ilustrados por quadrados, contendo três números e uma seta. O número que está na parte superior de cada genoma representa o número de inovação.

Outra propriedade do identificador de genes é a proteção de espécies. Com a informação de genes em mãos, é possível fazer a divisão de indivíduos em nichos. Isso permite que populações possam competir primariamente com indivíduos parecidos ao invés do todo, possibilitando que composição genética do nicho possa ser otimizada através da mutação e competição de aptidão (STANLEY; MIIKKULAINEN, 2002). O identificador permite também o cálculo da "distância evolutiva entre os genes", que é um medidor concreto de compatibilidade, representado na Equação 4.1.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W} \tag{4.1}$$

Depois de calculadas as distâncias, um limiar (*threshold*) é estabelecido para separar os indivíduos em espécies. A distância compatível entre os indivíduos é denotada por δ. O número de genes excessos é denotado por E. O número de genes disjuntos é denotado por D. A diferença de pesos entre genes correspondentes é denotada por $\overline{W}$. A importância de cada um desses 3 fatores é denotada por c_1, c_2, c_3.

Além disso, em uma rede neural, o identificador de genes também determina as posições das conexões de rede. Dependendo dessas posições, esses genes podem ganhar denominações especiais: **genes disjunto** e os **genes excesso**. Os genes disjuntos, assim como na Matemática, são genes que duas redes não possuem em comum. Os genes excesso também são disjuntos, mas são os genes de uma das redes que se situam no final de todos os outros genes. O gene excesso, disjunto e o identificador podem ser visualizados na Figura 4.2.

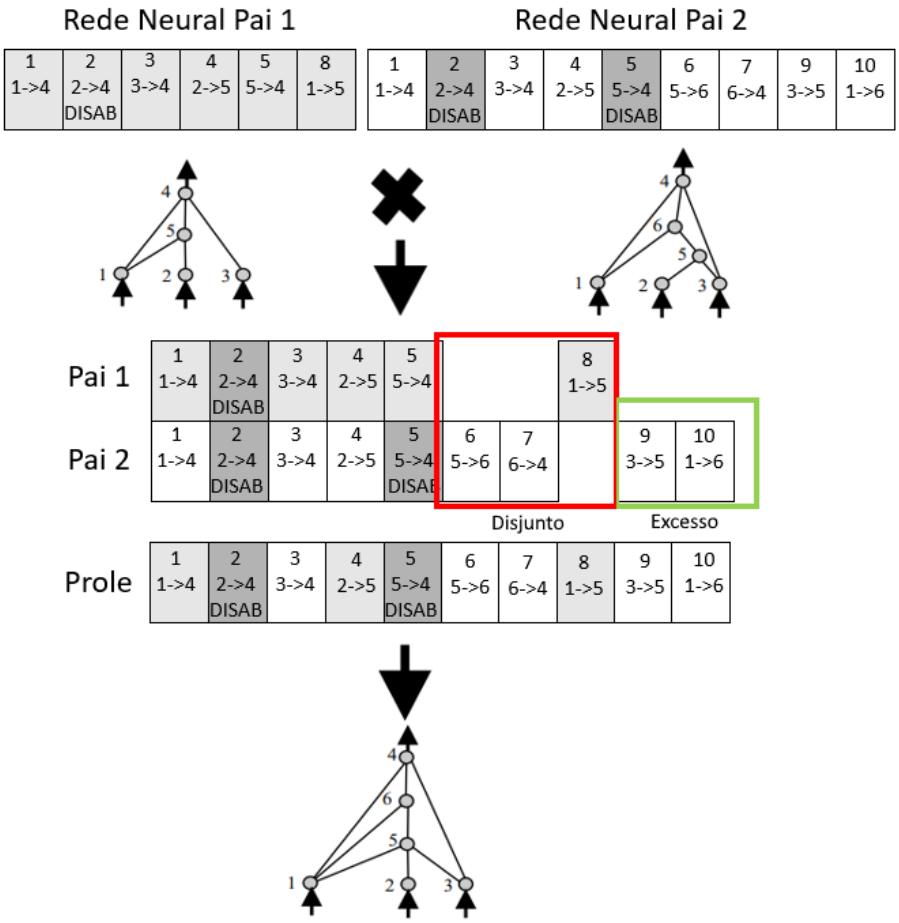

Figura 4.2: Imagem ilustrando o cruzamento de redes no algortimo NEAT (STANLEY; MIIKKULAINEN, 2002).

A Figura 4.2 mostra o cruzamento de duas redes distintas e como seus genomas, ilustrados por quadrados, estão arranjados. Na figura, o identificador dos genes, localizado ao topo de cada quadrado, indica os tipos de conexões existentes na rede neural. Por exemplo, se ambas as redes-pai possuem a conexão 1-4, ambas possuem conexão com mesmo identificador. No entanto, se uma rede possuir conexão que a outra não tem, eles serão genes disjuntos, como, por exemplo, a conexão de identificador 6 à 8 do pai 2 (sinalizados pelo indicador vermelho). Os genes excesso, demonstrados pelos genes de identificadores 9 e 10, são aqueles que são disjuntos, porém ficam em um dos pais no final do vetor de genes (sinalizados pelo indicador verde).

A Figura 4.2 também apresenta a ferramenta de cruzamento de redes. Na figura, existem dois pais diferentes, no entanto, a prole consegue ser gerada com sucesso graças à identificação de genes. Com o identificador, é possível verificar as conexões que os dois pais possuem em

comum, evitando conexões redundantes. Além disso, o identificador permitiu encontrar os genes disjuntos e excesso, que admitiu sua agregação à prole. É válido notar que os genes coincidentes são escolhidos dos pais mais aptos, enquanto os genes disjuntos e excesso são pegos de ambos os pais. Nesse caso, considerou-se que ambos pais possuíam mesmo desempenho, por isso, os genes foram escolhidos aleatoriamente.

Por último, a característica de aleatoriedade nas populações iniciais dos TWEANNs é descartada. Diferentemente dos outros algoritmos, NEAT inicia sua rotina de algoritmo com o total de zero nós ocultos (internos), começando com apenas a entrada e a saída do sistema. No método de incremento de estrutura do NEAT, novos nós e conexões surgem, através das mutações, e estruturas com pouco desempenho desaparecem, por meio do cálculo da função de aptidão (STANLEY; MIIKKULAINEN, 2002). Ou seja, as estruturas presentes nas evoluções são justificadas. Por fim, já que o algoritmo se inicia com um arranjo mínimo de topologia, a tendência é que o resultado final seja também minimizado, além de deixar o sistema menos complexo e com melhor desempenho.

4.4 CONCLUSÃO

Apesar de haver diversos métodos possíveis para realizar a neuro-evolução, uns mais simples e outros mais avançados, pode-se notar que a maioria deles possuem algum problema que os torna pouco atraentes para implementação, como, por exemplo, limitações de conexão ou o problema da permutação. Além de ser mais moderno que os outros algoritmos, as características que o algoritmo NEAT apresenta são interessantes para a implementação. Esses algoritmos e um resumo de suas características podem ser visualizados na Tabela 4.1.

Essencialmente, o algoritmo NEAT fez apenas algumas mudanças no projeto de um algoritmo de neuro-evolução, porém, essas modificações o tornaram um método mais robusto para a criação de redes neurais. Primeiro, a criação de um identificador de gene permitiu que conexões e características apresentadas pelas redes pudessem ser preservadas. Além disso, esse sistema tornou possível que o cruzamento de espécies ocorresse sem problemas. Esse algoritmo diminui o tamanho das estruturas possíveis ao descartar a iniciação aleatória. Por fim, através de

Tabela 4.1: Tabela de algoritmos citados neste trabalho.

Método	Codificação	Características
Structured Genetic Algorithm (sGA) (1993)	Direto, codificação binária	Conhecido pela simplicidade. Contém limitações. Mutações, reprodução, cruzamento e seleção.
NEAT (2002)	Direto, codificação por grafos	Implementa identificador de geração e inicialização vazia.
PDGP (1997)	Direto, codificação por grafos	Implementa cruzamento e mutação de subgrafos, inicialização de grafos aleatórios.
GNARL (1993)	Direto, codificação não-cruzada	Não possui cruzamento, enfatiza a mutação das espécies
Celullar Encoding (1993)	Indireto	Requer conhecimento aprofundado, estipula regras de criação, topologia indireta.

sua codificação, esse método desviou de vários problemas apresentados por outros algoritmos, limitação de conexão etc. Portanto, entre os algoritmos de codificação direta apresentados, o algoritmo NEAT foi escolhido para ser implementado.

É válido citar que o autor do algoritmo NEAT, Kenneth O. Stanley, em conjunto com outros autores, desenvolveu iterações e outras versões do algoritmo, o algoritmo HyperNEAT (GAUCI; STANLEY, 2007) e o ES-HyperNEAT (RISI; STANLEY, 2012). Contudo, essas versões não são interessantes para os objetivos desse trabalho pelo fato de serem uma codificação indireta, mais complexa, e porque, essencialmente, esses algoritmos são uma versão diferente do NEAT, que já é um algoritmo bem consolidado.

Capítulo 5

PROPOSTA DE ARQUITETURA

O objetivo principal desse trabalho é o desenvolvimento de uma ferramenta para auxiliar profissionais de saúde a avaliarem e decidirem sobre a adequabilidade de jogos digitais no auxílio do tratamento de pacientes. Para isso essa ferramenta deve ser capaz de autonomamente evoluir ao jogar um jogo e prover dados que possam ser usados na avaliação da dificuldade do mesmo. Assim, a ferramenta deve estar apta a:

- **Treinar redes neurais através de neuro-evolução**: Para o funcionamento correto da ferramenta e da aplicação, uma rede neural deve ser treinada utilizando técnicas de neuro-evolução. Para tal, informações do jogo devem ser coletadas e uma interface deve ser construída para que haja interação. Isso é um dos principais itens do trabalho.

- **Ser capaz de terminar a tarefa autonomamente**: No final do treinamento da rede neural, o modelo deve ser capaz de resolver o problema de maneira independente. É valido mencionar que a definição de problema no contexto de jogos é totalmente dependente do usuário que está treinando o modelo. Por exemplo, se uma rede neural foi treinada para atingir apenas o ponto que marca a metade do nível de um jogo, o modelo apenas será capaz de fazer isso.

- **Disponibilizar a rede final para uso**: Disponibilizar aos usuários uma forma de interagir com a rede neural treinada. Depois de construída a rede, o modelo será disposto ao usuário que queira utilizá-lo.

5.1 ARQUITETURA

A Figura 5.1 apresenta um fluxo de ações dentro da aplicação principal. O fluxograma apresenta os elementos que serão utilizados, construídos e gerados para a automação do jogo, terminando com um bloco de FIM, indicando o término do experimento.

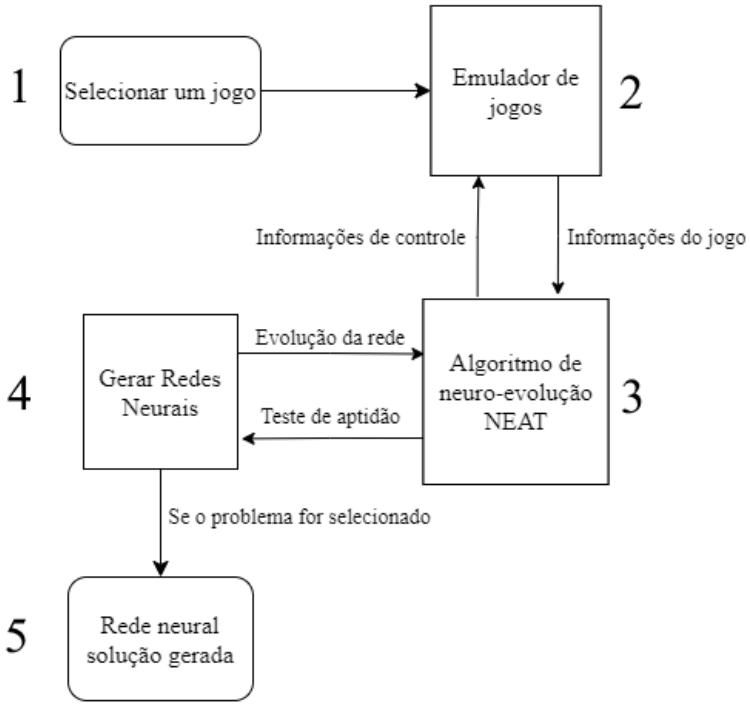

Figura 5.1: Fluxo de ações do trabalho nas aplicações principais

No início do processo, indicado por 1, um jogo deverá ser selecionado para a implementação do trabalho. Depois de escolhido, um emulador de jogos, indicado por 2, será utilizado para executar o jogo e fazer uma interface com o mesmo, coletando os dados de personagem, inimigos, terreno etc. Essas informações serão utilizadas pelo algoritmo de neuro-evolução NEAT, indicado por 3, para criar uma rede neural, indicada por 4. Essa rede neural terá seu desempenho medido depois de testado dentro do jogo. Para testar, o emulador dará acesso aos controles do jogo, permitindo que os algoritmos possam ser executados e utilizados no controle. Assim que avaliado, se o objetivo não for atingido, a rede será modificada e evoluída para que novas soluções possam surgir. Quando um resultado satisfatório for obtido, essa rede resultante será utilizada para jogar o jogo automaticamente, indicado por 5, finalizando o experimento.

A Figura 5.2 apresenta quais as informações que a interface de jogos extrai do jogo selecionado para enviar para o algoritmo de neuro-evolução.

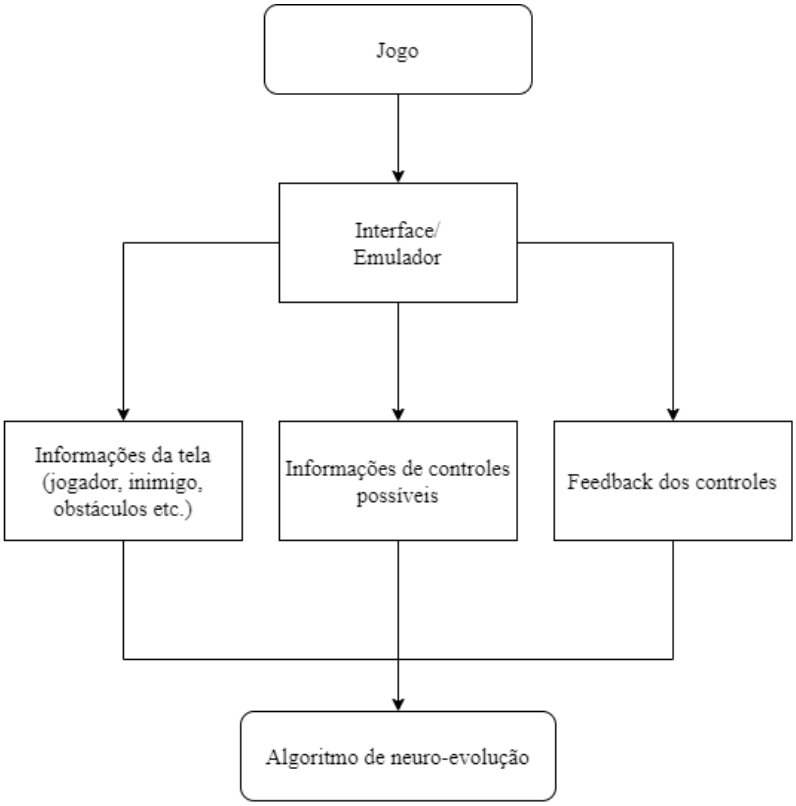

Figura 5.2: Fluxo de informações adquiridas do jogo

O emulador de jogos extrai do jogo: as informações da tela (dados de posição do jogador, inimigos, terreno etc.), as informações dos controles possíveis (no caso, os direcionais, botão de pular etc.) e os movimentos e interações que o usuário desempenha dentro do jogo. Todas essas informações serão enviadas para o algoritmo de neuro-evolução para serem utilizadas como entrada na rede neural. Esse esquema de transferência de dados de entrada, redes neurais e controles de saída podem ser vistas na Figura 5.3.

A Figura 5.3 ilustra como o fluxo de informações ocorrerá. O emulador, indicado pelo bloco de *input*, irá fornecer para a rede neural as informações relevantes para o movimento, como, por exemplo, a posição do personagem, inimigo, blocos que são possíveis atravessar (e.g. ar) e blocos que são possíveis pisar (e.g. chão). Em seguida, a rede neural a ser treinada usará essas informações como uma matriz de entrada. Utilizando neuro-evolução, essa rede será treinada e evoluída. Para testar a rede neural, os nós de saídas da rede serão conectados

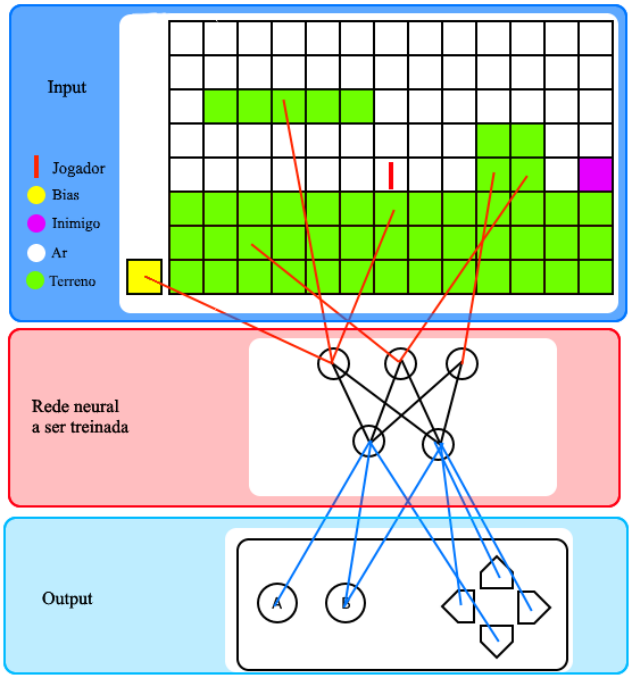

Figura 5.3: Ilustração do esquema de informações, controles e neuro-evolução a ser utilizada.

ao controle do jogo. Portanto, quando acontecer mudanças no jogo através do controle, as entradas consequentemente mudarão. Eventualmente, a rede neural será complexa o suficiente para interpretar tudo que ocorre na matriz de entrada e responder devidamente, terminando o jogo.

Capítulo 6

EXPERIMENTOS E RESULTADOS

Depois da arquitetura do projeto ter sido definida, os próximos passos são sua implementação, configuração e execução. As próximas seções detalham os tipos de plataforma utilizados, os métodos de implementação, escolhas de projeto e os resultados das execuções.

6.1 FERRAMENTAS, LINGUAGENS E RECURSOS UTILIZADOS

Para a realização desse trabalho, foi feito o uso de linguagens de programação, emuladores e ambientes de desenvolvimento. Os seguintes recursos foram utilizados:

- **Emulador de jogos**: Construído para emular jogos e controlá-los quadro a quadro, um emulador de jogos foi utilizado. Ele é capaz de:
  - Construir interfaces.
  - Coletar informações do jogo, como, por exemplo, dados de posição do personagem, inimigo etc.
  - Controlar o jogo por meio de programação.

  Unindo essa ferramenta com o algoritmo de neuro-evolução, a construção de um modelo que consiga jogar jogos foi possível.

- **Jogo a ser emulado, Super Mario Bros.**: Jogo eletrônico de plataforma, desenvolvido e publicado pela empresa japonesa *Nintendo*® em 1985 para o console *Famicon*® (WIKIPEDIA, 2021b). O console foi renomeado para *NES*®, *Nintendo Entertainment System*, quando lançado nos EUA. O jogo *Super Mario Bros.* (abreviado como SMB), mais especificamente o nível 1-1, primeira fase do jogo, foi escolhido para ser testado nesse trabalho pelos seguintes motivos:
  - Controles simples: Para controlar este jogo, são necessários os direcionais, um botão para pular e outro botão correr.
  - Jogabilidade desafiadora: SMB e outros jogos da década de 90 são conhecidos pela sua dificuldade elevada. Além disso, o gênero de jogos de plataforma requerem um melhor reflexo por parte do usuário, elemento que diretamente influencia a dificuldade e jogabilidade do jogo.
  - Objetivos simples: O objetivo desse jogo é claro, andar para frente e chegar ao final do nível, que pode ser indicado por uma bandeira ou um machado.
  - Arquivos leves: O arquivo de jogo do SMB é notoriamente conhecido pela sua leveza, advindo de tempos em que não havia muita memória computacional. O jogo *Super Mario Bros.* ocupa apenas 40KB de memória. Esse aspecto, além de ajudar no desempenho do sistema como um todo, torna o jogo fácil de ser reiniciado, parado e manipulado sem queda de quadros.
  - Documentado: Um dos maiores atrativos do SMB para a comunidade é sua popularidade. Por ser um artefato famoso e ser um jogo de 1985, o jogo é incrivelmente bem documentado. Existe uma página de *Internet* documentando o que cada endereço de memória significa (AUTHORS, 2005). Esse aspecto torna o jogo um ambiente de fácil trabalho para a produção de uma rede neural.
- **Linguagem de programação, Lua**: Projetada e desenvolvida pelos brasileiros Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes na Pontifícia Universidade Católica do Rio de Janeiro, LUA é uma linguagem de programação cujas vantagens são,

entre outros componentes, sua eficiência, clareza e facilidade de aprendizado. Graças a isso, LUA é utilizada em diversas aplicações de programação para controlar robôs, inteligências artificiais, processar texto, automatizar etc.

- **Ambiente de programação, Visual Studio**: O ambiente escolhido é, especificamente, o *Visual Studio Code, Community Edition* (MICROSOFT, 2015). Ele foi selecionado pela sua simplicidade de programação e integração com outras ferramentas, o que facilita o desenvolvimento.

- **Algoritmo de neuro-evolução, NEAT**: O algoritmo escolhido para fazer o treinamento neuro-evolutivo e a construção de uma rede neural, é o *NeuroEvolution of Augmenting Topologies (NEAT)* de Stanley e Miikkulainen. Ele foi escolhido pelo fato de ser um algoritmo testado, consolidado, evoluído de outras abordagens e minimalista (HEIDENREICH, 2019; STANLEY, 2015; PARKER; BRYANT, 2008).

6.2 EXTRAÇÃO DE CARACTERÍSTICAS DO PROBLEMA

Durante o processo de definição dos *inputs* desse sistema, houve um estudo abrangente da extração de dados e endereços de RAM do jogo escolhido, *Super Mario Bros.* Utilizando glossários de endereço disponíveis em sítios Web e a ferramenta de monitoramento de RAM do emulador de jogos *BizHawk* (TASVIDEOS, 2017), pôde-se chegar às informações da Tabela 6.1.

Essas informações foram utilizadas da seguinte maneira:

- **Endereços 0x0086 e 0x006D**: Utilizados juntos para determinar a posição de eixo **X** do jogador.

- **Endereço 0x03B8**: Utilizado para determinar a posição de eixo **Y** do jogador.

- **Endereço 0x00B5**: Informação do número de tela no eixo **Y**. Originalmente seria utilizado para movimentação vertical, no entanto, esse endereço é muito utilizado para determinar mortes em buracos dentro do jogo.

Tabela 6.1: Tabela de informações de RAM, do jogo *Super Mario Bros.*.

Endereço RAM	Informações
0x0086	**JOGADOR**: Informação da posição de eixo **x** na tela atual.
0x006D	**JOGADOR**: Informação da tela atual em que se encontra, no sentido do eixo **x**.
0x03B8	**JOGADOR**: Informação do eixo **y** na tela atual.
0x00B5	**JOGADOR**: Informação da tela atual no eixo **y**.
0x000E	**JOGADOR**: Informações de estado (Poderes, subindo vinhas, entrando em canos, morte, etc.).
0x001D	**JOGADOR**: Informação de "flutuação" do jogador.
0x0500-0x069F	**MAPA**: Informações dos ***tiles*** **(casas/blocos)** presentes no mapa, se eles são sólidos em que o jogador pode andar ou se são ar em que pode-se passar sobre. Não contêm informações relacionadas à textura.
0x000F-0x0013	**INIMIGOS**: Informações dos inimigos presentes na tela.
0x0087-0x008B	**INIMIGOS**: Esses endereços contêm informações de eixo **x** de 5 inimigos.
0x006E-0x0072	**INIMIGOS**: Esses endereços contêm informações da tela atual em que os inimigos se encontram.
0x00CF-0x00D3	**INIMIGOS**: Esses endereços contêm informações de eixo **y** de 5 inimigos.

- **Endereço 0x000E**: Informações de estado do jogador. Dentro do sistema, utilizado para determinar se o jogador morreu por inimigos ou por tempo.

- **Endereço 0x001D**: Informações relacionados a flutuação do jogador. Utilizado para detectar se o jogador está no "ar" ou na "terra". Também utilizado para determinar se o mastro de bandeira (a linha de chegada) foi tocada, cumprindo o objetivo do nível.

- **Endereços 0x0500 até 0x069F**: Esses endereços possuem as informações de terreno próximo ao jogador (chão, buraco, blocos etc.). Essa é uma informação crucial para a determinação do *input*.

- **Endereços 0x000F até 0x0013**: Esses cinco endereços armazenam a informação de presença de inimigos. Não contêm informações do inimigo, esse endereço meramente indica se eles existem ou não.

- **Endereços 0x0087 até 0x008B / 0x006E até 0x0072**: Analogamente às informações do jogador, esses cinco pares de endereços determinam a localização de eixo **x** dos 5 inimigos.

- **Endereços 0x00CF até 0x00D3**: Analogamente, esses endereços possuem as informações

de eixo **y** dos cinco inimigos.

Depois de extrair os dados de terreno, inimigos e jogador a partir da memória RAM do jogo, as informações foram rearranjadas para serem mostradas em um "mini-mapa" simplificado, assim, mostrando todas as informações relevantes presentes na tela (Figura 6.1).

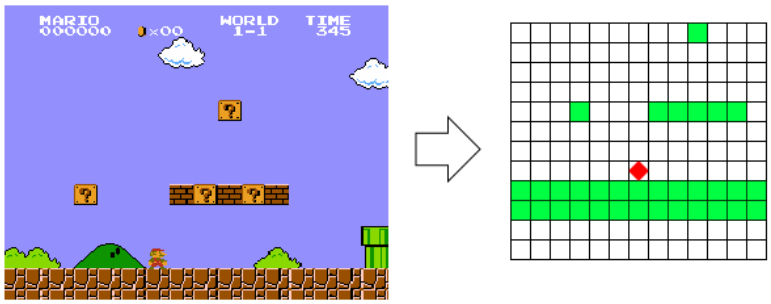

Figura 6.1: Esquema de conversão de dados do jogo em uma matriz de entrada.

Dessa maneira, determinou-se que essa matriz de entrada, ou representação simplificada da tela do usuário, tivesse um tamanho de 13x13 blocos, contendo um total de 169 caixas. Somando-se com o *input* de *bias* (Seção 2.3), o número de entradas para a neuro-evolução é 170.

O número 13 para a quantidade de linhas e colunas da matriz não foi escolhido de maneira arbitrária. Dentro do jogo *Super Mario Bros.*, existe um endereço de RAM (**0x0755**) que determina a distância em **pixels** que o jogador pode estar do canto esquerdo da tela. Em média, esse valor fica em torno de 112 **pixels**, equivalendo a 7 blocos dentro do jogo. Logo, para estimular uma margem de erro e tentar limitar a visão da rede, foi determinado que a matriz seria composta de 6 blocos para cada direção cardeal, totalizando 13x13 blocos.

Por fim, um último bloco relacionado ao *bias* foi introduzido aos *inputs*. Esse bloco permite que haja uma mudança na funções de ativação, ao se adicionar uma constante à entrada. Um neurônio pode ser constantemente ativado independente do que estiver ocorrendo na matriz de entrada (MARSLAND, 2011).

6.3 ANÁLISE E NEURO-EVOLUÇÃO

Depois de se obter os dados a partir dos endereços de memória e arranjá-los em uma matriz de entradas, o algoritmo de neuro-evolução NEAT foi aplicado para que as redes neurais pudessem ser geradas.

6.3.1 NEAT

Para criar o código de neuro-evolução, foram utilizados passos parecidos aos do Algoritmo 1, presente na Seção 2.4.1. Então, o Algoritmo 2 foi utilizado.

Algoritmo 2 Pseudo-código para o algoritmo NEAT utilizado na implementação, retirado de (STANLEY; MIIKKULAINEN, 2002)

```
INICIALIZAR população de redes neurais com topologias aleatórias mínimas;
DIVIDIR a população em espécies;
nGeracoes = 0;
enquanto (CHEGAR À LINHA DE CHEGADA) não é satisfeito faça
    AVALIAR todas redes neurais da população;
    SELECIONAR as melhores redes;
    RECOMBINAR genética com os selecionados;
    MUTAÇÃO da prole resultante;
    DIVIDIR a prole em espécies;
    COMEÇAR análise da nova população;
    nGeracoes = nGeracoes + 1;
fim
```

No começo da rotina, uma população de indivíduos foi iniciada. De acordo com o algoritmo NEAT, essas redes iniciais são mínimas, possuindo apenas entradas e saídas. No entanto, essas redes são mutadas uma vez, dando origem à alguns neurônios escondidos aleatórios e conexões aleatórias.

Antes de prosseguir, o algoritmo NEAT dita que a população deve ser dividida em espécies. Isso é necessário para que possa haver competição apenas entre redes similares e possibilitar formação de diferentes abordagens ao problema (STANLEY; MIIKKULAINEN, 2002). A equação utilizada para determinar a similaridade de espécies é a Equação 4.1 (Seção 4.3). Em seguida, o algoritmo é executado continuamente até que uma rede consiga alcançar o objetivo proposto do nível.

Depois de avaliar o *fitness* desempenhado, ou seja, a distância percorrida para a direita dentro do jogo, de todos os indivíduos da população, a rede de melhor desempenho de cada espécie é selecionada, por consequência, todas as redes não selecionadas dentro das espécies serão prontamente descartadas. Além disso, se a melhor rede não tiver seu desempenho melhorado através de algumas gerações, ele também será descartado. Em seguida, restando apenas a rede de melhor desempenho de cada espécie, o passo de recombinação inicia.

As redes neurais são escolhidas aleatoriamente e cruzadas umas às outras. Dessa maneira, serão geradas novas redes que possuem características (pesos, neurônios escondidos, conexões) dos indivíduos-pais. Pelo fato dos pais serem as redes de melhor desempenho de cada espécie, teoricamente o cruzamento deles geraria um indivíduo potencialmente melhor ainda. Após a produção de indivíduos suficientes para popular o grupo de redes, cada um desses seres serão mutados, modificando sua genética e topologia. Posteriormente, essa prole novamente será dividida em espécies, utilizando a equação de distância genética. Se o valor dessa equação não satisfizer o limiar (*threshold*) de proximidade estabelecido, essas redes podem criar uma nova espécie. Se houver adjacência, elas serão aglutinadas em alguma espécie já existente.

Por fim, depois de todo esse processo, uma nova geração é iniciada, dando início à uma nova rodada de evolução.

6.3.2 CONFIGURAÇÕES DO NEAT

Os hiper-parâmetros utilizados para o NEAT, que são as configurações utilizadas pelo algoritmo antes de iniciar o treino, foram valores obtidos de artigos (FRANÇA, 2018; STANLEY; BRYANT; MIIKKULAINEN, 2005; STANLEY; MIIKKULAINEN, 2004). Esses hiper-parâmetros são apresentados na Tabela 6.2.

Não foi definido um número máximo de gerações para o experimento. Além disso, o NEAT não estabelece um número máximo de neurônios e conexões escondidas que podem existir dentro das redes (STANLEY; MIIKKULAINEN, 2002). Por isso, foi estipulado um número grande para a quantidade de nós escondidos, um milhão. Por último, a função de ativação utilizada foi uma função sigmoide (MIRA; SANDOVAL, 1995) modificada para que os valores estejam

dentro do intervalo [-1, 1].

Tabela 6.2: Tabela de hiper-parâmetros utilizado para a neuro-evolução.

Hiper-parâmetros	Valores
População	300
Limite de proximidade genética	1.0
Coeficiente de disjunção	2.0
Coeficiente de pesos	1.0
Probabilidade mutação de pesos	0.4
Probabilidade adição de conexões	1.8
Probabilidade adição de nós	0.4
Probabilidade adição de conexões ao bias	0.35
Probabilidade de habilitar conexões	0.3
Probabilidade de desabilitar conexões	0.15
Probabilidade de cruzamento	0.9

Devido ao tamanho da matriz de entrada, alguns hiper-parâmetros tiveram que ser modificados. Em geral, todos os valores tiveram seus números aumentados para comportar a grande gama de possibilidades de conexões. A única mudança notória é a de probabilidade de conexão, que agora garante a adição de uma conexão e tem possibilidade de adicionar uma segunda.

6.3.3 MÉTODOS DE APROVAÇÃO

Dentro do processo de avaliação das redes, existe a ferramenta de recompensas que determina as melhores redes a serem selecionadas. Para isso, foi definido que o desempenho das redes seria definida pela posição **X** apresentada pelo personagem (endereços **0x0086** e **0x006D** da Tabela 6.1) subtraídos pela quantidade de quadros consumidos para a locomoção. Dessa maneira, a movimentação rápida é estimulada.

A cada quadro na emulação do jogo, um *script* de atualização do desempenho é executado. Se a posição **X** do jogador for maior que o anterior, a verificação daquela rede continua e a emulação prossegue até que um método de reprovação seja ativado. Dessa forma, a simulação de todas as redes dentro de todas as espécies recebem uma pontuação que é determinante para sua "sobrevivência" para a próxima geração.

Por fim, para que a rede final bem-sucedida seja destacada entre as outras redes de mesma

geração, definiu-se que seria adicionado 1000 de desempenho à pontuação caso tocasse no objetivo do nível (a bandeira). Nesse caso, esse aumento na pontuação de desempenho seria provocado pela mudança do endereço **0x001D**, indicando que o jogador tocou na linha de chegada.

6.3.4 MÉTODOS DE REPROVAÇÃO

Utilizando as informações obtidas através das memórias RAM, é possível medir parâmetros que reprovem o desempenho de uma rede neural.

No contexto do jogo *Super Mario Bros.*, o mais desejado é que o jogador consiga chegar o mais rápido possível à linha de chegada. Para isso, o usuário não pode ser morto ou ficar parado por muito tempo em um mesmo local. Por esses motivos, métodos para reprovar redes existem.

- ***Timeout***: Foi determinado um tempo limite pelo qual uma rede pode permanecer parada. Se o algoritmo detectar que jogador não se moveu por muito tempo, a avaliação da rede é imediatamente parada e passada para a próxima rede a ser testada. Esse tempo limite leva em consideração um tempo constante de 0.5 segundo e uma fração do tempo que o jogador permaneceu vivo. Dessa maneira, a movimentação de redes com bons desempenhos é estimulada, proporcionando mais oportunidades de movimento.

- **Morte por queda (0x00B5)**: Toda vez que uma morte por queda ocorrer, a rede neural para de ser testada e a próxima rede é executada.

- **Morte por inimigos ou tempo (0x000E)**: Analogamente à morte por queda, esse tipo de morte encerra a testagem da rede.

6.4 ANÁLISE DAS SAÍDAS PRODUZIDAS PELAS REDES

Logo após a lógica de neuro-evolução ser definida é preciso determinar os comandos de controle do jogo, ou o *output*, para que possa haver movimentação e aumento da posição **X**, que determina

a desempenho do indivíduo da rede e possibilita a neuro-evolução.

6.4.1 CONTROLES POSSÍVEIS

Primeiramente, foram determinados os comandos possíveis para o jogo. Existem oito botões disponíveis para o *NES*®, console do jogo escolhido:

- Botão A
- Botão B
- Botão de seta para cima
- Botão de seta para baixo
- Botão de seta para direita
- Botão de seta para esquerda
- Botão de *Select*
- Botão de *Start*

Desses oito botões, seis deles trazem um impacto direto para a jogabilidade. O botão **A** (pular), o **B** (andar mais rápido), e as direções de movimento (**direita**, **esquerda**, **cima** e **baixo**); os botões *Select* e *Start* são apenas utilizados para selecionar e mover-se no menu principal, portanto, serão desconsiderados.

6.4.2 MOVIMENTAÇÃO E FREQUÊNCIA

Depois de definir os controles possíveis, é preciso aplicar o pressionamento de botões ao jogo, ilustrados na parte de *outputs* da Figura 5.3.

O emulador escolhido para o trabalho, Bizhawk (TASVIDEOS, 2017), trabalha executando código LUA a cada quadro (*frame*) que se passa no jogo. Além disso, o console *NES*® normalmente tem seus jogos, incluindo o *Super Mario Bros.*, executados em 60 QPS (ORLAND, 2019). Dessa maneira, o emulador executa os *scripts* escritos a uma taxa de 60 vezes por segundo.

A partir de experimentos empíricos, descobriu-se que executar o algoritmo e a mudança de controle/movimentação em todos os 60 quadros torna o experimento lento e ineficiente. Por isso, foi regulado que a avaliação da rede neural e a consequente definição de botões para os movimentos seriam feitas em uma frequência de 12 vezes por segundo, ou a cada 5 quadros, equivalendo a $\frac{1}{12}$ de um segundo.

6.5 MÉTODO DE AVALIAÇÃO

Depois do jogo ter sido concluído com sucesso, alguns parâmetros de avaliação foram utilizados para medir a dificuldade do jogo, sendo elas:

- **Botões pressionados**: Quantidade de vezes que os botões foram pressionados durante o nível. Para tentar simular o pressionamento de botões de um ser humano, foi estabelecido que eles seriam contados a cada 12 quadros ($\frac{1}{5}$ de um segundo), o que estaria de acordo com o tempo de reação médio de um ser humano (WONG; HAITH; KRAKAUER, 2015).
- **Quantidade de inimigos**: Inimigos presentes na tela. Para determinar o número de inimigos presentes, foi contabilizado o número oponentes existentes na memória a cada 60 quadros (1 segundo).
- **Tempo dispendido**: Tempo necessário, em segundos, para concluir o nível (do início ao fim). Para estar de acordo com o QPS do jogo, um segundo foi medido a cada 60 quadros.

Assim que todos esses parâmetros forem contados, no final do nível, uma equação é feita:

$$NivelDificuldade = QtBotoes * a + Tempo * b + QtInimigos * c \tag{6.1}$$

Na Equação 6.1, foi estabelecido que o nível de dificuldade é determinado pelos parâmetros coletados multiplicados pelos valores a, b e c, onde $\{(a, b, c) \in \mathbb{R} | 0 \geq (a, b, c) \geq 1\}$.

Para esse trabalho, os valores das variáveis de ponderação foram determinados arbitrariamente, sendo eles $a = 0.6, b = 0.3, c = 0.1$. Desse modo, essa equação disponibiliza um número

concreto para determinar a dificuldade do jogo. Com essa quantificação, por exemplo, o profissional de saúde pode fazer comparações de jogos e avaliar o grau de dificuldade do jogo. Caso esse profissional necessite avaliar o jogo com parâmetros mais relacionados a botões, ele pode diminuir a relevância do tempo e quantidade de inimigos. Analogamente, ele pode estipular um valor mais alto para o tempo e para a quantidade de inimigos e assim por diante.

6.6 CONFIGURAÇÕES E TESTES REALIZADOS

No geral, não houve problemas na execução do algoritmo propriamente dito. No entanto, alguns efeitos colaterais ocorreram durante estágios do treinamento das redes neurais. Esses empecilhos não foram impeditivos para a evolução das redes, porém, de certa forma, eles afetaram a velocidade do treinamento.

Durante a execução do algoritmo, por exemplo, observou-se que as redes neurais eram incapazes de utilizar mecanismos de atalho presentes no jogo. No jogo *Super Mario Bros.*, existem certos pontos no mapa que são capazes de teletransportar o jogador e podem ser utilizados para terminar o nível mais rapidamente. Contudo, a rede neural não conseguia entrar nesse atalho por dois motivos principais: a rede fazia o jogador se movimentar continuamente, atrapalhando a entrada, e o tempo que era gasto para entrar no cano disparava o *timeout*, fazendo o algoritmo considerar a rede como imprópria. Apesar de ser um problema que seja impeditivo para o uso de um dos mecanismos do jogo, esse problema foi deixado de lado, porque o seu tratamento exige a adição de exceções, que aumentaria a complexidade do código, e porque esse problema não impede o nível de ser completado.

Além disso, durante o experimento, a rede encontrou uma maneira de contornar o *timeout* estabelecido. No decorrer da execução, por vezes, a rede neural sendo testada fazia o jogador pular no mesmo lugar. Normalmente isso faria o método de reprovação *timeout* ser executado. Porém, a rede neural estava movimentando o protagonista em um ritmo tão lento que o método de reprovação não estava sendo ativado. Para contornar isso, foi decidido que a rede necessitaria de uma movimentação significativa para não ser reprovada.

Por fim, para ilustrar a rede neural sendo testada e melhorar a visualização, o algoritmo

foi programado para desenhar uma figura contendo a entrada, rede neural e saída. Apesar de ser uma boa ilustração e ajudar no desenvolvimento do projeto, notou-se que essa ferramenta causava uma grande lentidão à execução. Por causa disso, durante boa parte do experimento, esse desenho dos componentes da rede neural foi desativado. Além disso, usou-se um recurso da ferramenta *Bizhawk* para acelerar a velocidade de execução. Dentro do emulador, existe uma ferramenta que destrava os QPS do jogo, dessa maneira, o emulador não é confinado a apenas 60 QPS. Graças a essa funcionalidade, foi possível aumentar a velocidade de execução de 60 QPS para 120 QPS, efetivamente dobrando a velocidade do experimento. Apesar de dobrar a velocidade, a execução não foi afetada, pois o *script* é executado apenas em quadros (frames) do jogo e não em segundos reais.

6.7 RESULTADO FINAL

Conforme o apresentado nas seções anteriores, o experimento foi executado e a rede neural foi treinada. No final do treinamento, foi verificado que o tempo total de execução foi de 70 horas, mesmo com o uso da aceleração de velocidade da ferramenta de emulação. Por fim, uma rede neural resposta foi gerada, ilustrada na Figura 6.2.

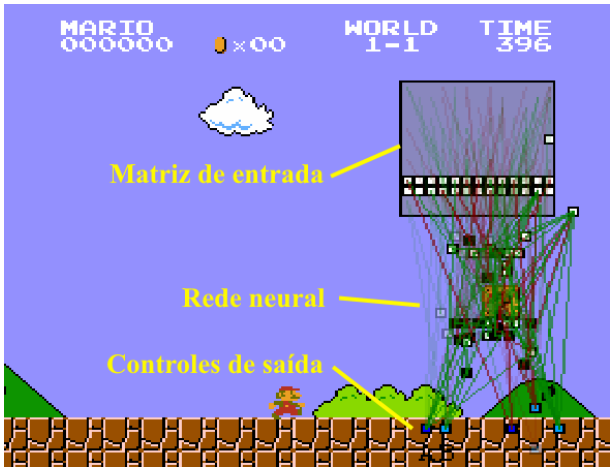

Figura 6.2: Ilustração da rede neural resultante do treinamento.

Na Figura 6.2, é possível ver a rede neural resultante. Nela, é possível ver, de cima para baixo, a matriz de entrada, a rede neural e os controles de saída. Na matriz de entrada, existe

um mini-mapa que representa o nível sendo jogado. Na rede neural, há um amontoado de nós e conexões que formam essa rede, as cores determinam os pesos das conexões e a ativação dos nós. Nos controles de saída, estão presentes os seis controles possíveis para o jogo.

Na rede neural solução, houve um total de 170 neurônios de entrada, 45 escondidos e 6 de saída. Foram geradas 8 camadas para os neurônios escondidos. Por fim, a função de ativação utilizada foi a função sigmoide (Seção 6.3.2). As informações de camada da rede estão presentes na Tabela 6.3.

Tabela 6.3: Camadas existentes na Rede Neural Solução produzida

Camada	Número de neurônios
Entrada	170
1º escondida	22
2º escondida	10
3º escondida	4
4º escondida	3
5º escondida	2
6º escondida	2
7º escondida	1
8º escondida	1
Saída	6

A Figura 6.3 apresenta um gráfico da evolução de desempenho (ou a posição **X** do jogador) ao longo de gerações. Observando esse gráfico, é possível notar que houve gargalos na melhora de desempenho em alguns momentos. Por exemplo, as gerações 31 a 51 e 76 a 106 foram períodos em que as redes neurais permaneceram estagnadas. Eventualmente a neuro-evolução conseguiu evoluir as redes e encontrar as soluções que resolvessem esses gargalos, chegando ao final da fase. Por fim, a subida repentina à direita do gráfico é explicada pela mecânica de ganho de desempenho ao finalizar o jogo. Nessa mecânica, quando o jogador toca a linha de chegada, ele ganha 1000 de desempenho.

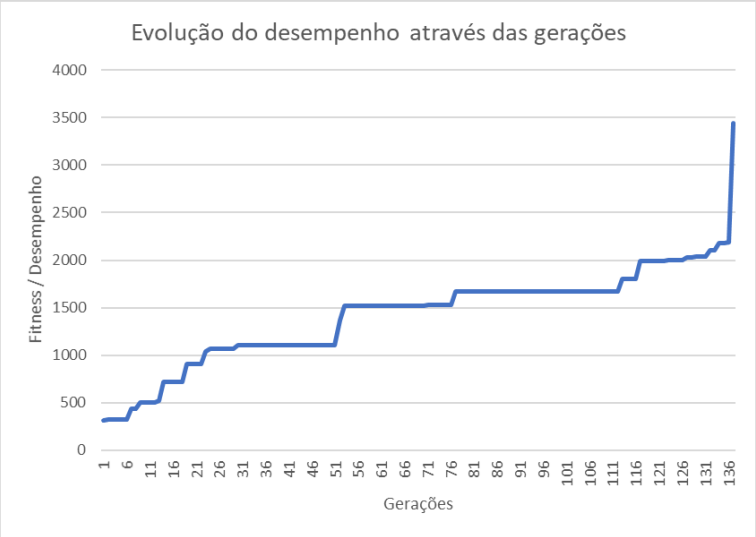

Figura 6.3: A evolução de fitness através das gerações de redes.

6.8 RESULTADO DA AVALIAÇÃO

Durante a execução das redes, foi programado que elas guardassem os valores que seriam relevantes para a avaliação de dificuldade: o número de botões pressionados, o número de inimigos presentes na tela a cada segundo e o tempo de conclusão do nível. Contudo, através dos experimentos empíricos, notou-se que esses valores não possuíam indicativos de evolução, pois, mesmo que uma rede execute todas as ações da maneira mais ineficiente possível, se ela possuir a maior pontuação (*fitness*) da população, ela ainda seria passada para a próxima geração. Isso acontece devido às funcionalidades evolutivas da neuro-evolução. Por causa desse problema, foi determinado que apenas a rede final seria avaliada e não haveria necessidade dos gráficos dos botões, inimigos e tempo.

A Tabela 6.4 apresenta os valores que compõem a avaliação e a quantificação da dificuldade da rede neural final.

Tabela 6.4: Informações de avaliação da rede-solução.

Informação	Quantidade
Número de botões	408
Tempo para concluir	23
Número de inimigos por segundo	105
Dificuldade	268.1

Utilizando a Equação 6.1 da Seção 6.5, foi possível agregar o número de botões, tempo de conclusão e número de inimigos em apenas um único número: a dificuldade. Se os profissionais de saúde utilizarem esse número para comparar com outro jogo que eles já utilizam, é possível fazer uma avaliação prévia e superficial desse novo jogo. Dessa maneira, esses profissionais podem otimizar seu tempo ao aferir a adequabilidade daquele jogo.

6.9 ANÁLISE DE EVOLUÇÃO DAS REDES

Durante a execução das redes, houve mecanismos de evolução que permitiram a melhora das redes e evitaram gargalos. As Figuras 6.4 e 6.5 se situam em diferentes gerações do experimento, porém, eles apresentam ferramentas de neuro-evolução que o NEAT aplica. Uma dessas ferramentas é a evolução gradual de rede. Essa melhora através da topologia é mostrada na Figura 6.4.

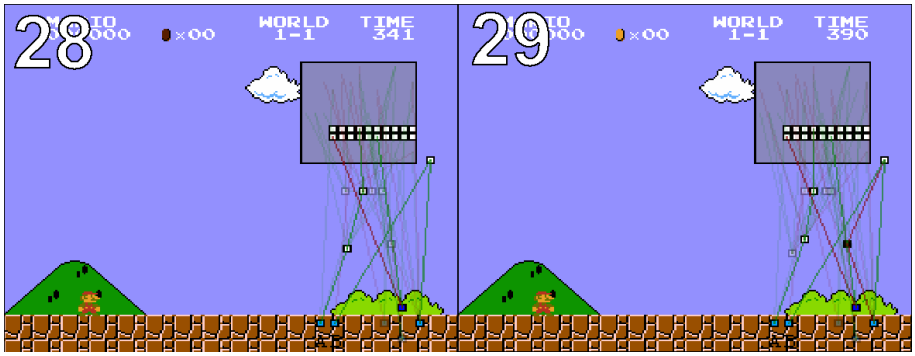

Figura 6.4: Ilustração da rede de melhor desempenho das gerações 28 e 29.

Essa figura apresenta duas gerações de rede neural. Mais especificamente, essas redes são as de melhor desempenho da geração 28 e da 29 do treinamento, as gerações são indicadas no canto superior esquerdo da figura. Para medir o *fitness*, essas redes foram executadas. Quando houve uma eventual estagnação dessas redes, seu desempenho foi medido considerando o tempo e distância percorrida. Portanto, essas gerações possuem *fitness* igual a 1071 e 1109, respectivamente, indicando que houve uma melhora de desempenho. Para fazer essa melhora, a neuro-evolução fez mudanças de topologia na rede neural. Na Figura 6.4, pode-se notar que houve a adição de alguns nós e algumas conexões novas à rede. Graças a essas adições, a

rede neural, quando testada, percorreu um caminho levemente diferente dentro do jogo e o seu desempenho foi melhorado.

Além de existir a evolução através da adição de conteúdo às redes, é possível também ter evolução através da diferenciação de espécies e abordagens. Esse tipo de evolução está ilustrado na Figura 6.5. Nessa figura, estão ilustradas as melhores redes das gerações 50 a 52. Essas gerações estão indicadas por números no canto superior esquerdo das imagens. Quando medidos, o *fitness* dessas três redes são 1109, 1368 e 1523.5, respectivamente.

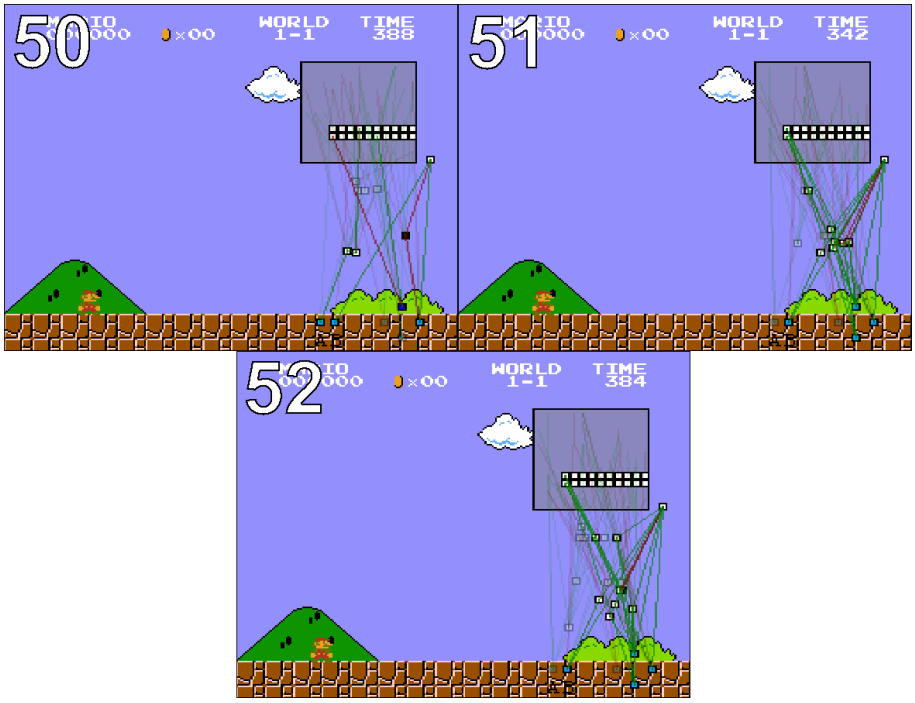

Figura 6.5: Ilustração da rede de melhor desempenho das gerações 50 à 52.

A Figura 6.5 exibe como pode haver um aumento de desempenho ao fazer a troca de espécies. Na figura, é possível observar que, na geração 50, a rede de melhor *fitness* é completamente diferente da rede de melhor desempenho da geração seguinte. Nesse caso, ao invés de ocorrer uma adição de nós e conexões à rede para melhorar o desempenho, houve uma substituição. Já que a abordagem apresentada por uma outra espécie (rede neural) conseguiu superar o desempenho da rede de melhor *fitness* atual, a abordagem alternativa que desempenhou melhor foi a escolhida para representar a geração. Isso ocorre graças à capacidade do algoritmo NEAT de separar essas redes em espécies (STANLEY; MIIKKULAINEN, 2002).

Por fim, a geração 52 é um outro exemplo de evolução através da topologia, que já foi

demonstrada na Figura 6.4. Pode-se notar que são grandes as semelhanças entre as redes das gerações 51 e 52. Portanto, houve evolução de desempenho ao se adicionar elementos à rede neural.

6.10 CONCLUSÃO

Durante o experimento houveram algumas complicações mínimas, no entanto, o trabalho foi concluído de maneira satisfatória. Através das configurações e testes, observou-se que a neuro-evolução consegue achar problemas imprevisíveis quando foi capaz de contornar a ferramenta de *Timeout* dos métodos de reprovação. Porém, a neuro-evolução também consegue contornar obstáculos do mapa que causavam gargalos ao desempenho, otimizando e melhorando a evolução das redes neurais.

Por fim, um sistema de avaliação utilizando parâmetros externos ao jogo foi criado. Por exemplo, houve a utilização do número de botões, tempo de conclusão da fase e o número de inimigos presentes na tela. Esse sistema, apesar de ser arbitrário, pode ser usado como uma ferramenta de comparação para avaliar jogos. Dessa maneira, ele consegue auxiliar os profissionais de saúde a aferirem a adequabilidade do jogo.

Capítulo 7

CONSIDERAÇÕES FINAIS

O uso de neuro-evolução em jogos na Literatura tem crescido e se desenvolvido rapidamente (RISI; TOGELIUS, 2017). Múltiplos trabalhos que utilizam NEAT foram encontrados, no entanto, não se encontrou muitas produções que trabalhassem o aspecto de avaliação de jogos.

No decorrer do trabalho, observou-se que a neuro-evolução consegue de maneira eficiente construir redes neurais. Utilizando os recursos de evolução e divisão de espécies do NEAT (HEIDENREICH, 2019), atingiu-se um ponto em que esse tipo de método criou diferentes tipos de soluções e contornou problemas que eventualmente poderiam ter ficado estagnados.

Além disso, nesse trabalho, graças aos métodos de Stanley (STANLEY; MIIKKULAINEN, 2002), não foi preciso jogar os jogos previamente para ter uma coleta de dados. Mesmo assim, houve um desenvolvimento satisfatório durante o qual as redes neurais evoluíram e criaram soluções ao problema apresentado. Devido ao grande número de implementações existentes, encontrar referências de programação foi fácil, no entanto, o número de programações em LUA disponíveis ainda continua pequeno, o que é justificado pela sua baixa popularidade.

Por fim, a criação do sistema de pontuação de dificuldade de jogos desse trabalho é apenas uma prova de conceito para o potencial de ampliação da utilização de jogos na reabilitação de pacientes, utilizando parâmetros pouco considerados nos jogos. Esse tipo de medição necessita de mais experimentos até que ele seja utilizada em contextos reais com profissionais de saúde e pacientes.

7.1 TRABALHOS FUTUROS

Ainda que esse trabalho tenha obtido resultados aceitáveis, há certos pontos que poderiam ter sido aperfeiçoados e modificados.

Uma das questões é o número de execuções. Nesse trabalho, executou-se o treinamento de rede neural apenas uma vez. De acordo com a estatística, para se entender o comportamento da população, é necessário ter um certo número de amostragem das execuções (MORETTIN, 2010). Alguns autores, em seus problemas de otimização, executam e testam redes neurais 5000 vezes para conseguir gerar amostras e analisá-las (JONG, 2002). Por essa razão, o número de execuções e amostragem feita nesse trabalho é impróprio para análise. Em um trabalho futuro o algoritmo de neuro-evolução será executado um número maior de vezes de modo a criar milhares de redes neurais diferentes para que a análise estatística do mesmo seja realizada.

Analogamente à otimização, um tema relacionado é a execução de treinamentos com diferentes hiper-parâmetros. Essa questão está relacionada ao número de execuções, pois, no problema de otimização, além de procurar a melhor rede, os hiper-parâmetros modificam os tipos de rede resultantes do treinamento. Por essa razão, para se otimizar essa rede-neural-solução é necessário testar vários hiper-parâmetros diferentes, milhares de vezes cada vez.

Outro problema que pode ser tratado no futuro é a questão de níveis do jogo. Esse trabalho focou apenas no treinamento e resolução do primeiro nível do jogo *Super Mario Bros.* Entretanto, trabalhar múltiplos níveis seria o caminho natural. Fazendo isso, poder-se-ia criar uma rede neural que consiga jogar o jogo por inteiro e, desse modo, avaliar o jogo em sua completude.

Outro mecanismo que não foi desenvolvido é o de teletransporte, presente no jogo escolhido. No jogo *Super Mario Bros.*, existem canos verdes que podem levar o jogador a outros mapas, dependendo de como é usado, esses mapas alternativos podem ser usados como atalho para finalizar um mapa mais rapidamente. Esse problema foi citado na Seção 6.6. Por isso, em um trabalho futuro, será considerado esse tipo de mecanismo e atalho do jogo, caso exista a necessidade de finalizar o nível o mais rápido possível.

Assim como a neuro-evolução trabalhou no jogo *Super Mario Bros.* nesse trabalho, outros jogos podem ser avaliados utilizando o mesmo método. Títulos como *Super Metroid*, *Super*

Mario Land, *Super Mario Bros 3* e *Sonic CD* são exemplos de jogos de plataforma que podem ser testados.

Semelhantemente, jogos de diferentes naturezas e gêneros também podem ser testados. Títulos como *Super Mario 64* e *Kirby 64: Crystal Shard* são exemplos de jogos de plataforma em 3D que podem ser utilizados. Em outros gêneros, existem jogos de tiro de primeira pessoa como *Wolfenstein* e *Goldeneye*.

Durante o desenvolvimento do código, foi de muita ajuda o fato de o jogo escolhido possuir uma vasta documentação. Se um jogo menos conhecido tivesse sido selecionado, provavelmente não seria tão fácil conseguir as posições dos inimigos, do mapa ou do jogador.

Por último, o uso do algoritmo de neuro-evolução NEAT para o treinamento resultou em redes neurais satisfatórias. No entanto, para próximos trabalhos, também pode se tentar usar outros algoritmos genéticos, como o HyperNEAT (GAUCI; STANLEY, 2007).

Referências Bibliográficas

ABANDONWARE, M. *CHESSMASTER 8000*. 2000. Disponível em: <https://www.myabandonware.com/game/chessmaster-8000-doa>. Acesso em: 3 de Março de 2022.

ABSTRACTS. *Biometrics*, [Wiley, International Biometric Society], v. 21, n. 3, p. 761–777, 1965. ISSN 0006341X, 15410420. Disponível em: <http://www.jstor.org/stable/2528559>.

ANGELINE, P. J. et al. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 1994.

APONTE, M.-V.; LEVIEUX, G.; NATKIN, S. Measuring the level of difficulty in single player video games. *Entertainment Computing*, v. 2, 01 2009.

AUTHORS, S. *Super Mario Bros.:RAM map*. 2005. Disponível em: <https://datacrystal.romhacking.net/wiki/Super_Mario_Bros.:RAM_map>. Acesso em: 20 de dezembro de 2021.

BALLARD CLIVE G.; CORBETT, A. C. H. O. A. Can brain training games improve cognition in peope over 60? *Alzheimer's & Dementia*, Elsevier Science, v. 6, p. e55–e56, 2010. ISSN 1552-5260. Disponível em: <http://doi.org/10.1016/j.jalz.2010.08.171>.

BARROSO, S. M. et al. Treinamento cognitivo de atenção e memória de universitários com jogos eletrônicos. *Psico*, v. 50, n. 4, p. e29466–e29466, 2019.

BERNER, C. et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

BORG, P. J. F. G. I. *Modern Multidimensional Scaling: Theory and Applications*. 2nd. ed. [S.l.]: Springer, 2005. (Springer Series in Statistics). ISBN 9780387251509,0387251502.

BRAUN, H.; WEISBROD, J. Evolving neural feedforward networks. 1993.

BRINK JOSEPH RICHARDS, M. F. H. *Real-World Machine Learning*. 1. ed. Manning Publications, 2016. ISBN 1617291927,9781617291920. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=f22a7835b960bceb3b248a89c864da48>.

BRO, R.; SMILDE, A. K. Principal component analysis. *Analytical methods*, Royal society of chemistry, v. 6, n. 9, p. 2812–2831, 2014.

CARDOSO, N. d. O.; LANDENBERGER, T.; ARGIMON, I. I. de L. Jogos eletrônicos como instrumentos de intervenção no declínio cognitivo–uma revisão sistemática. *Revista de Psicologia da IMED*, Faculdade Meridional-IMED, v. 9, n. 1, p. 119–139, 2017.

CAZELLA, S. C.; NUNES, M.; REATEGUI, E. A ciência da opinião: Estado da arte em sistemas de recomendação. *André Ponce de Leon F. de Carvalho; Tomasz Kowaltowski..(Org.). Jornada de Atualização de Informática-JAI*, p. 161–216, 2010.

CFA, I. *Conheça as quatro Revolucoes Industriais que moldaram a trajetória do mundo*. 2019. Disponível em: <https://cfa.org.br/as-outras-revolucoes-industriais/>. Acesso em: 3 de Março de 2022.

CHANDRA, A. L. *McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron*. 2018. Disponível em: <towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1>. Acesso em: 10 de dezembro de 2021.

CLUNE, J. et al. On the performance of indirect encoding across the continuum of regularity. *IEEE Transactions on Evolutionary Computation*, IEEE, v. 15, n. 3, p. 346–367, 2011.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.

DASGUPTA, D.; MCGREGOR, D. R. Designing application-specific neural networks using the structured genetic algorithm. p. 87–96, 1992.

DASGUPTA, D.; MCGREGOR, D. R. *sGA: A structured genetic algorithm*. [S.l.]: Citeseer, 1993.

DUCOSIM. *Ducosim*. 2020. Disponível em: <https://www.ducosim.nl>. Acesso em: 3 de Março de 2022.

EIBEN, A. E.; SMITH, J. E. et al. *Introduction to evolutionary computing*. [S.l.]: Springer, 2003. v. 53.

EIBEN, J. S. a. A. Introduction to evolutionary computing. *Natural Computing Series*, Springer-Verlag Berlin Heidelberg, 2015.

EIBEN, J. S. A. E. *Introduction to Evolutionary Computing*. [S.l.]: Springer, 2008. (Natural computing series). ISBN 9783540401841; 3540401849.

EXAME, E. E. *Pesquisa aponta que 3 em cada 4 brasileiros jogam jogos eletrônicos*. 2020. Disponível em: <https://exame.com/colunistas/esporte-executivo/pesquisa-aponta-que-3-em-cada-4-brasileiros-jogam-jogos-eletronicos/>. Acesso em: 3 de Março de 2022.

FLOREANO, D.; DURR, P.; MATTIUSSI, C. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, v. 1, n. 1, p. 47–62, 2008.

FOGEL, D. B. et al. A self-learning evolutionary chess program. *Proceedings of the IEEE*, IEEE, v. 92, n. 12, p. 1947–1954, 2004.

FRANÇA, L. D. R. Neuroevolution of augmenting topologies applied to the detection of cancer in medical images. 2018.

GAUCI, J.; STANLEY, K. Generating large-scale neural networks through discovering geometric regularities. In: *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. [S.l.: s.n.], 2007. p. 997–1004.

GOMEZ, F.; MIIKKULAINEN, R. Incremental evolution of complex general behavior. *Adaptive Behavior*, Sage Publications Sage CA: Thousand Oaks, CA, v. 5, n. 3-4, p. 317–342, 1997.

GRAND, S. et al. Creatures: Artificial life autonomous software agents for home entertainment. *Proceedings of the first international conference on Autonomous agents*, p. 22–29, 01 1997.

GRUAU, F. Neural network synthesis using cellular encoding and the genetic algorithm. Citeseer, 1994.

HANSEN, N.; OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, MIT Press, v. 9, n. 2, p. 159–195, 2001.

HAUSKNECHT MATTHEW; LEHMAN, J. M. R. S. P. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 6, p. 355–366, 2014. ISSN 1943-068X,1943-0698. Disponível em: <http://doi.org/10.1109/TCIAIG.2013.2294713>.

HEIDENREICH, H. *NEAT: An Awesome Approach to NeuroEvolution*. 2019. Disponível em: <https://towardsdatascience.com/neat-an-awesome-approach-to-neuroevolution-3eca5cc7930f>. Acesso em: 26 de novembro de 2021.

HO, T. K. Random decision forests. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. [S.l.: s.n.], 1995. v. 1, p. 278–282 vol.1.

HOLLAND, J. H. Genetic algorithms. *Scholarpedia*, v. 7, n. 12, p. 1482, 2012. Revision #128222.

HUANG, T. Computer Vision: Evolution And Promise. 1996. Disponível em: <http://cds.cern.ch/record/400313>.

HURWITZ, J.; KIRSCH, D. *Machine Learning*. [S.l.]: IBM, 2018.

HURWITZ, J.; KIRSCH, D. Machine learning for dummies. *IBM Limited Edition*, John Wiley & Sons, Inc, v. 75, 2018.

IBMCLOUD. *Machine Learning*. 2020. Disponível em: <https://www.ibm.com/cloud/learn/machine-learning>. Acesso em: 10 de dezembro de 2021.

IBMCLOUD. *Machine Learning: Unsupervised Learning*. 2020. Disponível em: <https://www.ibm.com/cloud/learn/unsupervised-learning>. Acesso em: 10 de dezembro de 2021.

IZENMAN, A. J. Introduction to manifold learning. *Wiley Interdisciplinary Reviews: Computational Statistics*, Wiley Online Library, v. 4, n. 5, p. 439–446, 2012.

JALLOV, D.; RISI, S.; TOGELIUS, J. Evocommander: A novel game based on evolving and switching between artificial brains. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 9, n. 2, p. 181–191, 2016.

JONG, K. A. D. *Evolutionary computation: a unified approach*. 1st. ed. The MIT Press, 2002. ISBN 9780262041942,0262041944. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=0e8d3e34b41a8eebf70948d2e116a6ff>.

KI, H.-w.; LYU, J.-h.; OH, K.-s. Real-time neuroevolution to imitate a game player. In: PAN, Z. et al. (Ed.). *Technologies for E-Learning and Digital Entertainment*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 658–668. ISBN 978-3-540-33424-8.

KONGTHON, A. et al. Implementing an online help desk system based on conversational agent. In: *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*. New York, NY, USA: Association for Computing Machinery, 2009. (MEDES '09). ISBN 9781605588292. Disponível em: <https://doi.org/10.1145/1643823.1643908>.

LEHMAN, J.; MIIKKULAINEN, R. Neuroevolution. *Scholarpedia*, v. 8, n. 6, p. 30977, 2013. Revision #137053.

LOCKETT, A. J.; MIIKKULAINEN, R. Evolving opponent models for texas hold'em. In: IEEE. *2008 IEEE Symposium On Computational Intelligence and Games*. [S.l.], 2008. p. 31–38.

LOPEZ-SAMANIEGO, L. et al. Cognitive rehabilitation based on working brain reflexes using computer games over iPad. In: *2014 Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES)*. IEEE, 2014. Disponível em: <https://doi.org/10.1109\%2Fcgames.2014.6934155>.

LUGER, G. *Inteligência Artificial: Estruturas e estratégias para a solução de problemas complexos*. Bookman, 2004. ISBN 9788536303963. Disponível em: <https://books.google.com.br/books?id=ruZNPgAACAAJ>.

MAIMON, L. R. O. *Data Mining and Knowledge Discovery Handbook*. 1. ed. [S.l.]: Springer, 2005. ISBN 9780387244358,0387244352,9780387254654.

MANDISCHER, M. Representation and evolution of neural networks. p. 643–649, 1993.

MARSLAND, S. *Machine learning: an algorithmic perspective*. [S.l.]: Chapman and Hall/CRC, 2011.

MATOS, E. C. do A.; LIMA, M. A. S. Jogos eletrônicos e educação: Notas sobre a aprendizagem em ambientes interativos. *RENOTE*, v. 13, n. 1, 2015.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MICROSOFT. *Visual Studio Code*. 2015. Disponível em: <https://code.visualstudio.com/>. Acesso em: 3 de Março de 2022.

MICROSOFT. *XBOX*. 2021. Disponível em: <https://www.xbox.com/pt-BR/>. Acesso em: 02 de Dezembro de 2021.

MICROSOFT. *Xbox Adaptive Controller*. 2021. Disponível em: <https://www.xbox.com/en-CA/accessories/controllers/xbox-adaptive-controller>. Acesso em: 02 de Dezembro de 2021.

MIRA, J.; SANDOVAL, F. *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995: Proceedings*. [S.l.]: Springer Science & Business Media, 1995. v. 930.

MORAL, P. D.; MICLO, L. Branching and interacting particle systems. Approximations of Feynman-Kac formulae with applications to non-linear filtering. *Séminaire de probabilités de Strasbourg*, Springer - Lecture Notes in Mathematics, v. 34, p. 1–145, 2000. Disponível em: <http://archive.numdam.org/item/SPS_2000__34__1_0/>.

MORETTIN, L. G. *Estatística Básica - Probabilidade e Inferência*. Pearson Prentice Hall, 2010. (09-09445). ISBN 978-85-7605-370-5. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=f8fcb0e2eb17d30b4e908a6c52a4b9fc>.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. Madison, WI, USA: Omnipress, 2010. (ICML'10), p. 807–814. ISBN 9781605589077.

ORDONEZ TIAGO NASCIMENTO; BORGES, F. K. C. S. S. C. C. d. N. H. S. S. L.-S. T. B. Actively station: Effects on global cognition of mature adults and healthy elderly program using eletronic games. *Dementia & Neuropsychologia*, v. 11, p. 186–197, 2017. ISSN 1980-5764. Disponível em: <http://doi.org/10.1590/1980-57642016dn11-020011>.

ORLAND, K. *How an SNES emulator solved overclocking*. 2019. Disponível em: <https://arstechnica.com/gaming/2019/08/blast-processing-in-2019-how-an-snes-emulator-solved-overclocking/>. Acesso em: 3 de Março de 2022.

OTTERLO, M. V.; WIERING, M. Reinforcement learning and markov decision processes. In: *Reinforcement learning*. [S.l.]: Springer, 2012. p. 3–42.

PARKER, M.; BRYANT, B. D. Neuro-visual control in the quake II game engine. In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE, 2008. Disponível em: <Parker:2008>.

PEDERSEN C.; TOGELIUS, J. Y. G. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 2, p. 54–67, 2010. ISSN 1943-068X,1943-0698. Disponível em: <http://doi.org/10.1109/tciaig.2010.2043950>.

PEGI. *PEGI game rating*. 2020. Disponível em: <https://pegi.info/page/how-we-rate-games>. Acesso em: 3 de Março de 2022.

PEREZ-LIEBANA, D. et al. *General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms*. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1802.10363>.

PRESS SAUL A. TEUKOLSKY, W. T. V. B. P. F. W. H. *Numerical recipes: the art of scientific computing*. 3rd ed. ed. [S.l.]: Cambridge University Press, 2007. ISBN 9780511335556,9780521880688,0511335555,0521880688,9780521884075,0521884071,9780521706858,052170

PUJOL, J. C. F.; POLI, R. Evolving the topology and the weights of neural networks using a dual representation. Applied Intelligence, 1998.

RADCLIFFE, N. J. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing & Applications*, Springer, v. 1, n. 1, p. 67–90, 1993.

RISI, S. et al. Petalz: Search-based procedural content generation for the casual gamer. *IEEE Transactions on Computational Intelligence and AI in Games*, v. 8, n. 3, p. 244–255, 2016.

RISI, S.; STANLEY, K. O. An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density, and Connectivity of Neurons. *Artificial Life*, v. 18, n. 4, p. 331–363, 10 2012. ISSN 1064-5462. Disponível em: <https://doi.org/10.1162/ARTL_a_00071>.

RISI, S.; TOGELIUS, J. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, IEEE, v. 9, n. 1, p. 25–41, 2015.

RISI, S.; TOGELIUS, J. *Neuroevolution in Games: State of the Art and Open Challenges*. [S.l.], 2017. v. 9, n. 1, 25-41 p.

RONALD, E.; SCHOENAUER, M. Genetic lander: An experiment in accurate neuro-genetic control. Springer-Verlag, p. 452–461, 1994.

RUMMERY, G. A.; NIRANJAN, M. *On-line Q-learning using connectionist systems*. [S.l.]: Citeseer, 1994. v. 37.

RUSSELL, S.; NORVIG, P.; CANNY, J. *Artificial Intelligence: A Modern Approach*. Prentice Hall/Pearson Education, 2003. (Prentice Hall series in artificial intelligence). ISBN 9780137903955. Disponível em: <https://books.google.com.br/books?id=KI2WQgAACAAJ>.

SCHRUM, J.; MIIKKULAINEN, R. Evolving multimodal behavior with modular neural networks in ms. pac-man. In: *Proceedings of the 2014 annual conference on genetic and evolutionary computation*. [S.l.: s.n.], 2014. p. 325–332.

SIENA, M. C. d. S. et al. O uso de jogos digitais como ferramenta auxiliar no ensino da matemática e o protótipo do game sinapsis. Universidade Federal de Goiás, 2018.

SOARES, A. Y. et al. Os efeitos da prática de jogos eletrônicos ativos na qualidade de vida de pacientes com a doença de parkinson: uma revisão sistemática. Florianópolis, SC., 2017.

SOLARI, G. *Daltonismo, acessibilidade e o cara que joga "Zelda" sem enxergar*. 2011. Disponível em: <https://tecnologia.uol.com.br/ultnot/2011/04/27/alem-do-jogo-daltonismo-acessibilidade-zelda.jhtm>. Acesso em: 3 de Março de 2022.

SOUZA, S. F. de. *Uma análise da tradução oficial do jogo Life is Strange*. 2011. Disponível em: <https://monografias.brasilescola.uol.com.br/arte-cultura/traducao-localizacao-uma-analise-da-traducao-oficial-do-jogo-life-is-strange.htm>. Acesso em: 3 de Março de 2022.

STANLEY, K. *Find the Right Version of NEAT for Your Needs*. 2015. Disponível em: <http://eplex.cs.ucf.edu/neat_software/\#NEAT>. Acesso em: 27 de novembro de 2021.

STANLEY, K. O. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, Springer, v. 8, n. 2, p. 131–162, 2007.

STANLEY, K. O.; BRYANT, B. D.; MIIKKULAINEN, R. Real-time neuroevolution in the nero video game. *IEEE transactions on evolutionary computation*, IEEE, v. 9, n. 6, p. 653–668, 2005.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, v. 10, n. 2, p. 99–127, 2002. Disponível em: <http://nn.cs.utexas.edu/?stanley:ec02>.

STANLEY, K. O.; MIIKKULAINEN, R. Evolving a roving eye for go. In: SPRINGER. *Genetic and Evolutionary Computation Conference*. [S.l.], 2004. p. 1226–1238.

TASVIDEOS. *https://tasvideos.org/Bizhawk*. 2017. Disponível em: <https://tasvideos.org/Bizhawk>. Acesso em: 3 de Março de 2022.

TOLLES, J.; MEURER, W. J. Logistic Regression: Relating Patient Characteristics to Outcomes. *JAMA*, v. 316, n. 5, p. 533–534, 08 2016. ISSN 0098-7484. Disponível em: <https://doi.org/10.1001/jama.2016.7653>.

UNIVERSITY, I. S. *Video game ratings work, if you use them*. 2017. Disponível em: <https://www.sciencedaily.com/releases/2017/01/170125145805.htm>. Acesso em: 3 de Março de 2022.

WANG, W. *Machine Audition: Principles, Algorithms and Systems: Principles, Algorithms and Systems*. [S.l.]: IGI Global, 2010.

WATKINS, C. J.; DAYAN, P. Q-learning. *Machine learning*, Springer, v. 8, n. 3, p. 279–292, 1992.

WEBB, G. I.; KEOGH, E.; MIIKKULAINEN, R. Naïve bayes. *Encyclopedia of machine learning*, v. 15, p. 713–714, 2010.

WIKIPEDIA. *Sonic Adventure*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Sonic_Adventure>. Acesso em: 02 de Dezembro de 2021.

WIKIPEDIA. *Super Mario Bros.* 2021. Disponível em: <https://en.wikipedia.org/wiki/Super_Mario_Bros.> Acesso em: 02 de Dezembro de 2021.

WONG, A. L.; HAITH, A. M.; KRAKAUER, J. W. Motor planning. *The Neuroscientist*, Sage Publications Sage CA: Los Angeles, CA, v. 21, n. 4, p. 385–398, 2015.

YAN, X. G. S. X. *Linear Regression Analysis: Theory and Computing*. 1. ed. [S.l.]: World Scientific Publishing Company, 2009. ISBN 9789812834102,9812834109.

ZHU, D. *How I Built an Intelligent Agent to Play Flappy Bird*. 2020. Disponível em: <https://medium.com/analytics-vidhya/how-i-built-an-ai-to-play-flappy-bird-81b672b66521>. Acesso em: 13 de dezembro de 2021.