Universidade Federal Fluminense
Instituto de ciência e tecnologia
Curso de Graduação em Ciência da Computação

José Victor Dias Pereira

Avaliação da eficácia das Estratégias Evolutivas no treinamento de Redes Neurais Perceptron aplicadas a Jogos Digitais

Rio das Ostras
2023

José Victor Dias Pereira

Avaliação da eficácia das Estratégias Evolutivas no treinamento de Redes Neurais Perceptron aplicadas a Jogos Digitais

Trabalho de conclusão de curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto de Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Orientador: Prof. Dr. André Renato Villela da Silva

Rio das Ostras

2023

.

A figura referente ao arquivo

FichaCatalografica.jpg

fornecido pela Biblioteca

deverá aparecer aqui.

ATENÇÃO: Na versão impressa, essa página deverá ficar no verso da página anterior.

.

José Victor Dias Pereira

Avaliação da eficácia das Estratégias Evolutivas no treinamento de Redes Neurais Perceptron aplicadas a Jogos Digitais

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto de Ciência e Tecnologia da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel

Aprovada em 29 de 06 de 2023.

BANCA EXAMINADORA

Prof. Dr. André Renato Villela da Silva- Orientador
Universidade Federal Fluminese

Prof. Dra. Leila Weitzel Coelho da Silva

Prof. Dr. Marcos Ribeiro Quinet de Andrade

Rio das Ostras
2023

”Machine intelligence is the last invention that humanity will ever need to make.”
(Nick Bostrom)

Resumo

As Redes Neurais Artificiais são algoritmos amplamente referenciados na literatura e estão sendo progressivamente aplicados a problemas do mundo real. Por isso, encontrar métodos capazes de treiná-las de forma eficaz pode trazer benefícios para a sociedade. Além disso, os jogos digitais representam um desafio para os algoritmos de treinamento, por apresentarem ambientes complexos e dinâmicos. Contudo, um dos algoritmos de otimização capazes de lidar com ambientes complexos e dinâmicos são as Estratégias Evolutivas. Dessa forma, este trabalho se propõe a avaliar a eficácia das Estratégias Evolutivas quando utilizadas como método de treinamento para Redes Neurais Perceptron no contexto de jogos digitais. Para atingir este objetivo, os algoritmos e os jogos utilizados foram implementados e os treinamentos realizados. Após treinadas, as redes neurais apresentaram uma acurácia em torno de 94%, demonstrando assim, a efetividade do método nos jogos selecionados.

Palavras-chave: Redes Neurais Artificiais. Estratégias Evolutivas. Jogos Digitais. Aprendizado de Máquina. Algoritmos Evolutivos.

Abstract

Artificial Neural Networks are widely referenced algorithms in the literature and are progressively being applied to real-world problems. Therefore, finding methods capable of effectively training them can bring benefits to society. Additionally, digital games present a challenge for training algorithms due to their complex and dynamic environments. However, one of the optimization algorithms capable of dealing with complex and dynamic environments is Evolutionary Strategies. Thus, this study aims to evaluate the effectiveness of Evolutionary Strategies when used as a training method for Perceptron Neural Networks in the context of digital games. To achieve this objective, the algorithms and games used were implemented and the training was carried out. After being trained, the neural networks demonstrated an accuracy of around 94%, thus demonstrating the effectiveness of the method in the selected games.

Keywords: Artificial Neural Networks. Evolution Strategies. Digital Games. Machine Learning. Evolutionary Algorithms.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

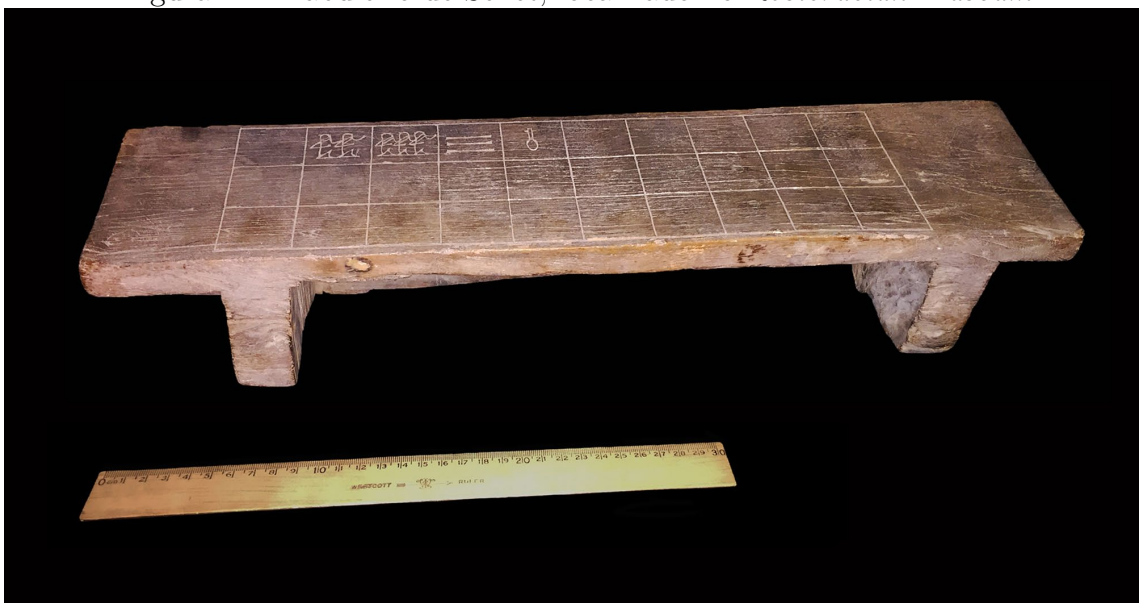
A palavra “jogo” é derivada de “jocus”, do latim, que significa gracejo, brincadeira ou divertimento[1]. Porém, Huizinga[2] propõe uma definição mais ampla, de que o jogo é um conceito primitivo, que surge antes da cultura humana, pois de acordo com o autor, a ideia de “jogo” também é observada em outras espécies de seres vivos, como por exemplo, os cães.

Os cães convidam uns aos outros para brincar utilizando um determinado protocolo com atitudes e gestos. Essa brincadeira é uma disputa que respeita algumas regras caninas, como por exemplo, não morder de verdade o outro. Essa disputa é considerada lúdica pois não é uma disputa real, mas sim, imaginada. Dessa forma, Huizinga[2] argumenta que um jogo é uma atividade lúdica mais abrangente que um mero fenômeno físico ou reflexo psicológico, sendo ainda uma de suas características o ato de, deliberadamente, evadir do mundo real.

De acordo com achados arqueológicos, um dos jogos mais antigos já criado pela humanidade é o Senet.[3] Senet é um jogo composto por um tabuleiro retangular contendo 30 casas e algumas peças. A figura 1.1 exibe um exemplar do tabuleiro. Esse jogo foi muito popular no Egito Antigo, por volta de 3000 a.C.[4]

Os detalhes das regras do jogo ainda são motivo de discussão, mas Piccione[4] afirma que: “As evidências disponíveis indicam que Senet era essencialmente um jogo de corrida, que envolvia posicionamento e estratégia. Era jogado por dois oponentes, cada um com seu conjunto de peças.” As regras do jogo foram modificadas pelos egípcios ao longo do tempo, mas sabe-se que ele foi jogado continuamente por mais de 3000 anos.[3]

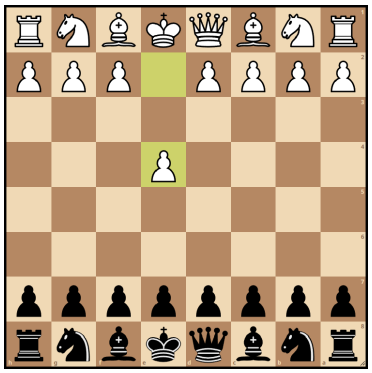
Figura 1.1: Tabuleiro de Senet, localizado no *Rosicrucian Museum*



Fonte: [5]

Com o avanço da tecnologia, a execução dos jogos se tornou gradativamente menos analógica e mais digital. Jogos que antes só podiam ser jogados com objetos físicos como tabuleiros, cartas e bolas, agora possuem versões digitais, desenvolvidas e executadas em computadores. A figura 1.2 exibe versões digitais de jogos tradicionais.

Figura 1.2: Versões digitais de jogos analógicos (xadrez, poker, futebol)



(a) Xadrez[1]

(b) Poker[2]

(c) Futebol[3]

Em 2020, a quantidade de pessoas que jogam regularmente algum tipo de jogo digital alcançou a marca de 3,1 bilhões, representando 40% da população mundial.[6] Além disso, em 2016, a indústria dos jogos faturou em torno de 91 bilhões de dólares,

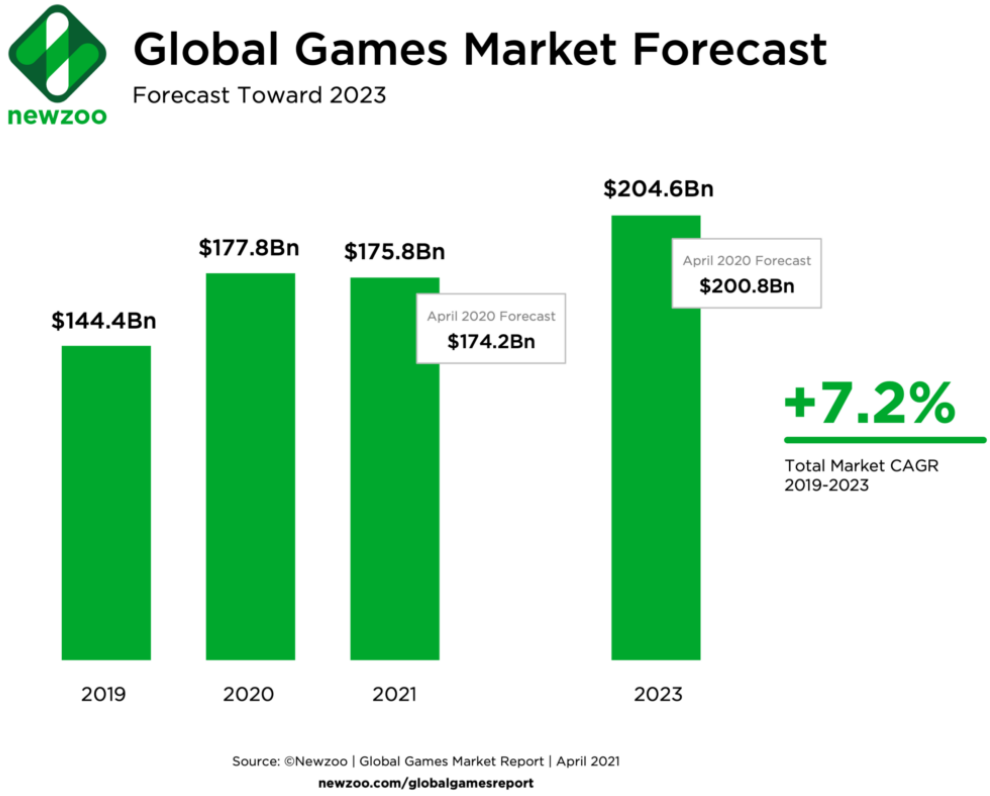
[1]Disponível em `www.lichess.org`

[2]Disponível em `www.jogatina.com/poker-online.html`

[3]Disponível em `www.ea.com/pt-br/games/fifa/fifa-21`

superando o cinema e a indústria fonográfica.[7] Mas em 2021, apenas 5 anos depois, esse faturamento quase duplicou, chegando ao valor de 175 bilhões de dólares e com projeções de crescimento para os próximos anos.[8] A figura 1.3 demonstra uma taxa de crescimento anual de 7,2% entre 2019 e 2023, ultrapassando os 200 bilhões de dólares no ano de 2023.

Figura 1.3: Projeção do faturamento da indústria de jogos para o ano 2023 (em dólares).

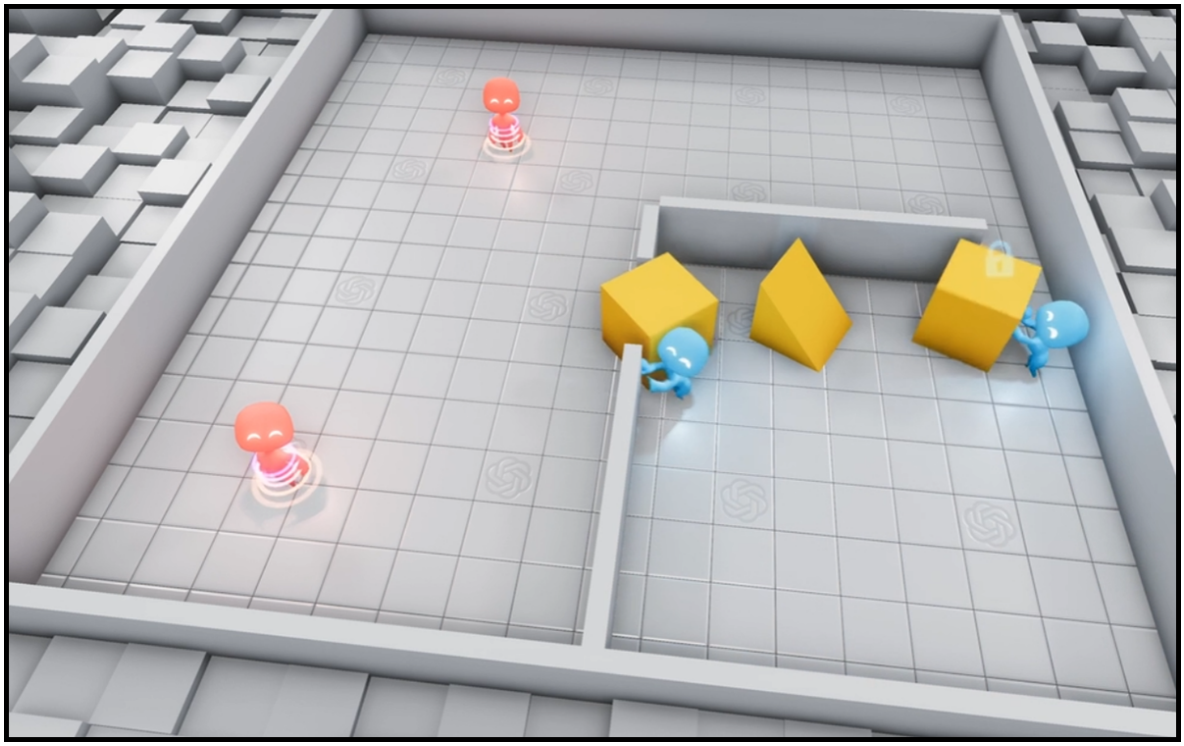


Fonte: [8]

Devido à digitalização dos jogos, suas aplicações avançaram além do espectro do entretenimento, possibilitando que sejam usados como ferramenta para outras situações, como por exemplo, na ciência.

Em 2020, a *OpenAI* publicou um trabalho onde um jogo simulando a popular brincadeira "pique-esconde" foi desenvolvido com o propósito de ser um ambiente de testes para algoritmos de Inteligência Artificial.[9] O objetivo do trabalho foi analisar as estratégias que os agentes desenvolveram para conseguir atingir seus respectivos objetivos na brincadeira. A figura 1.4 exibe uma estratégia desenvolvida pelos agentes.

Figura 1.4: O time azul aprendeu a usar objetos para se esconder do time vermelho.

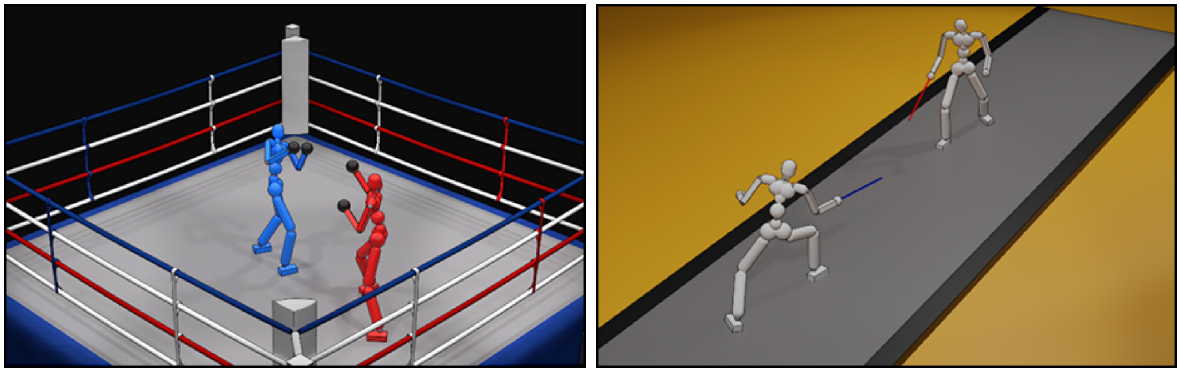


Fonte:[9]

Em 2021, pesquisadores do *Facebook AI Research* também desenvolveram dois jogos com a finalidade de avaliar algoritmos[10]. O objetivo do trabalho foi demonstrar a capacidade de aprendizagem do algoritmo proposto colocando-o para controlar personagens fisicamente simulados em um ambiente de competição esportiva.

A figura 1.5 exibe dois personagens disputando entre si enquanto se comportam de acordo com as estratégias aprendidas.

Figura 1.5: Jogos desenvolvidos pela equipe *Facebook AI Research*



(a) Boxe (b) Esgrima

Fonte:[10]

Na literatura é possível encontrar muitos jogos digitais desenvolvidos especificamente para serem usados como ambientes de testes[11], e geralmente com um objetivo em comum: analisar o desempenho ou o comportamento dos algoritmos atuando nesses cenários.

Utilizar os jogos como ambiente de avaliação possibilita que os pesquisadores testem hipóteses com alguns benefícios:

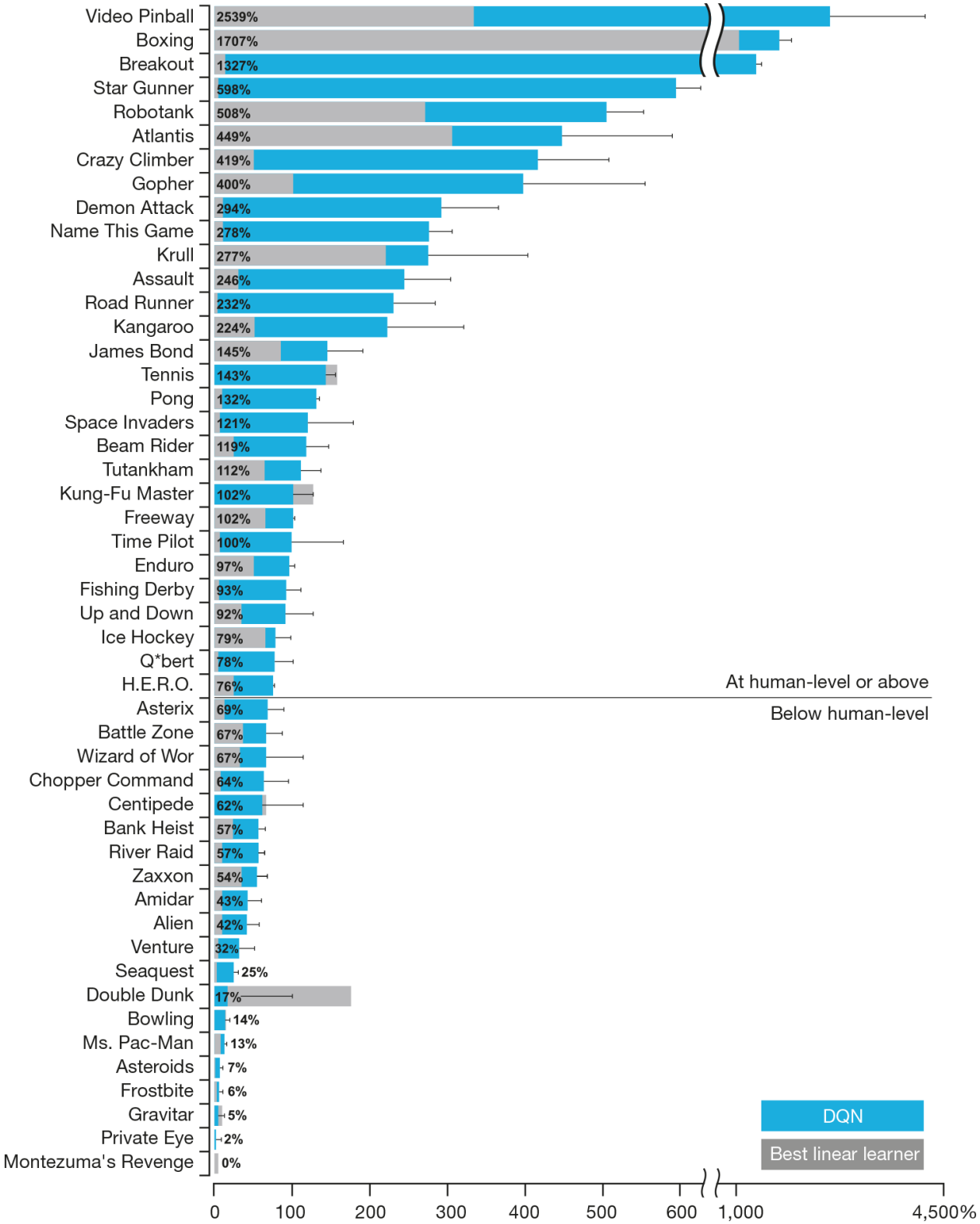
- Menor custo: Tendo em vista que os jogos digitais são ambientes virtuais, o risco de danificar objetos e dispositivos envolvidos no processo (robôs, veículos, placas, circuitos etc) é minimizado.

- Maior agilidade: Considerando o controle total do ambiente virtual por parte dos avaliadores, alterar ou reiniciar os cenários de testes torna-se mais simples em ambientes virtuais do que em cenários reais, composto por objetos reais.

- Maior inteligibilidade: Por serem sistemas visuais (feitos por elementos gráficos), analisar o comportamento dos algoritmos avaliados nesses ambientes torna-se uma tarefa mais inteligível/compreensível.

Além desses benefícios, os jogos digitais possuem uma característica notável: eles são sistemas complexos o suficiente para desafiar a capacidade de aprendizado dos algoritmos de Inteligência Artificial. Em 2015, os pesquisadores da *DeepMind* publicaram um trabalho onde Redes Neurais Artificiais conseguiram aprender a jogar vários jogos de um antigo videogame denominado “Atari”, utilizando aprendizado por reforço[12]. A figura 1.6 exibe o desempenho do algoritmo em cada jogo. De acordo com a figura, as redes neurais conseguiram atingir um desempenho sobre-humano em alguns jogos e sub-humano em outros.

Figura 1.6: Gráfico contendo o desempenho das Redes Neurais nos jogos do Atari. 0% representa a performance de um algoritmo que joga aleatoriamente enquanto 100% equivale a performance de um jogador humano profissional.



Fonte:[12]

As Redes Neurais Artificiais são modelos amplamente referenciados na literatura e vêm sendo progressivamente aplicados a problemas do mundo real. Por isso, descobrir métodos capazes de treiná-las de forma eficaz pode trazer benefícios para a sociedade. Além disso, as Estratégias Evolutivas são uma classe de algoritmos de otimização que possuem a capacidade de lidar com ambientes complexos e dinâmicos (assim como os jogos). Dessa forma, percebe-se a oportunidade de avaliar a eficácia das Estratégias Evolutivas como método de treinamento para as redes neurais artificiais, utilizando os jogos digitais como ambiente de avaliação.

1.1 Problema de Pesquisa

As Estratégias Evolutivas são eficazes como método de treinamento para Redes Neurais em jogos digitais?

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo geral deste trabalho consiste em avaliar a eficácia das Estratégias Evolutivas no treinamento de Redes Neurais Perceptron no contexto de jogos digitais.

1.2.2 Objetivos Específicos

Para atingir o objetivo geral foram definidos os seguintes objetivos específicos:

- Descrever os fundamentos e o funcionamento dos algoritmos utilizados;
- Implementar os algoritmos e os jogos selecionados;
- Realizar o treinamento das Redes Neurais e analisar os resultados;

1.3 Hipótese

Parte-se da hipótese de que as Redes Neurais Perceptron, quando treinadas com as Estratégias Evolutivas, conseguem atingir um alto desempenho nos jogos digitais selecionados, tendo em vista que, as Estratégias Evolutivas são algoritmos de otimização

estilo caixa-preta e isto as tornam métodos promissores para o ambiente complexo e não-supervisionado dos jogos digitais.

1.4 Organização do Trabalho

No Capítulo 2 é descrito os fundamentos e o funcionamento das Estratégias Evolutivas e das Redes Neurais Artificiais.

No Capítulo 3 é descrita a forma como foi realizada a implementação e a configuração das Estratégias Evolutivas, das Redes Neurais Perceptron, dos jogos selecionados e da interface que permite a comunicação entre todos esses elementos.

No Capítulo 4 é descrito como foram realizados os treinamentos das Redes Neurais e apresentados os resultados.

1.5 Prévia da Conclusão

Ao final, conclui-se que os objetivos são atingidos e a pergunta encontra-se respondida com a confirmação da hipótese, indicando que o método proposto obteve êxito no treinamento das redes neurais, apresentando elevado desempenho nos jogos implementados.

Capítulo 2

Fundamentação Teórica

Neste capítulo serão apresentados os fundamentos necessários para a compreensão dos algoritmos utilizados no trabalho. Ambos os algoritmos pertencem à Inteligência Computacional, um ramo de pesquisa da Inteligência Artificial.

2.1 Estratégias Evolutivas

As informações presentes nesta seção são referentes ao livro "*Manual de Computação Evolutiva e Metaheurística*" de Gaspar et al[13].

As Estratégias Evolutivas(EEs) são algoritmos de otimização baseados em heurísticas. Porém, para compreender inteiramente o seu funcionamento, se faz necessária uma breve introdução sobre o que é Otimização e o que são Heurísticas.

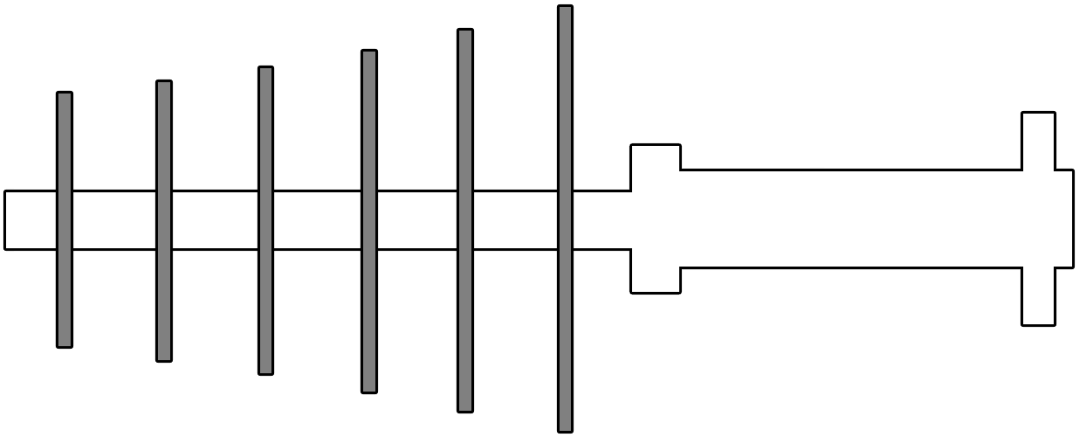
2.1.1 Otimização

Considere uma situação onde se queira construir uma antena. Por razões do domínio, a antena deve (obrigatoriamente) ser composta por seis elementos metálicos dispostos a uma certa distância uns dos outros, como mostra a figura 2.1.

É desejável que a eficiência energética da antena seja a maior possível. Porém, a eficiência energética desse tipo de antena depende dos comprimentos individuais de cada um dos elementos metálicos e das distâncias entre cada par deles. Por isto, a pergunta chave do problema é: Quais valores estes comprimentos e estas distâncias devem assumir para que a eficiência energética da antena seja máxima? Visando resolver esse tipo de

problema, estabeleceu-se a área de pesquisa denominada Otimização[1].

Figura 2.1: A antena. As barras cinzas verticais representam os elementos metálicos. Cada elemento tem seu próprio comprimento e uma distância ao elemento vizinho.



Fonte: O autor

De acordo com Gaspar[13]: "A Otimização é o campo de conhecimentos cujas técnicas visam determinar os extremos (máximos ou mínimos) de funções, em domínios determinados." Na prática, essas funções representam alguma característica relevante de um sistema real, enquanto os extremos, por sua vez, representam a configuração do sistema que maximiza ou minimiza tal característica.

Os elementos em um processo de otimização possuem algumas nomenclaturas que serão utilizadas ao longo deste trabalho:

- *função-objetivo*: a função cujo os extremos pretende-se determinar.
- *variáveis de decisão*: os argumentos(valores de entrada) da função-objetivo.
- *ótimo global*: o extremo buscado(ou máximo, ou mínimo).

No exemplo da antena, a eficiência energética está em função(depende) dos comprimentos e das distâncias dos elementos metálicos. Dessa maneira, tem-se um espaço vetorial onde cada ponto representa uma possível antena, com seu próprio valor de eficiência energética. Determinar os extremos da função significa buscar dentro desse espaço as antenas com a maior/menor eficiência energética. Porém, por este ser um exemplo de um problema de maximização, não se tem interesse na antena com a menor eficiência. Este exemplo possui algumas especificidades importantes:

[1]O termo "Otimização" (com inicial maiúscula) se refere ao campo de estudo, enquanto "otimização" (com inicial minúscula) se refere ao processo em si.

- A função-objetivo(eficiência energética) não possui uma expressão explícita, o cálculo do valor da função em um determinado ponto é feito através de uma simulação computacional.

- Realizar o cálculo da função-objetivo via simulação computacional envolve um processamento computacional custoso, pois o programa receberá os onze valores(seis comprimentos dos elementos metálicos e cinco distâncias entre eles) das variáveis de decisão e simulará a antena utilizando as leis da física, para só então, calcular a eficiência. Esse processo consome bastante tempo de CPU e por esse motivo é desejável que o processo de otimização realize este cálculo o mínimo possível.

- Considerando o fato da avaliação da função-objetivo ser computacionalmente custosa, a solução de se buscar a efetividade máxima da antena verificando todos os possíveis valores para cada uma das variáveis de decisão(busca exaustiva) se torna inviável, dado a quantidade exorbitante de tempo necessária para verificar todas as possibilidades.

- Partindo da premissa de que a antena não possuirá quilômetros de extensão, restrições podem ser adicionadas ao processo de otimização para limitar a área de busca, fazendo-o procurar possíveis antenas em uma região numérica realmente viável para a construção real da antena. A região numérica que contém apenas os valores válidos no sistema real é denominada *região factível* ou *conjunto factível*.

Diversas técnicas foram desenvolvidas com o objetivo de encontrar o ótimo global. Porém, algumas dessas técnicas só garantem essa localização em problemas com funções-objetivos em formatos específicos. Alguns exemplos são:

- Métodos de pontos interiores[14]: Quando a função objetivo é convexa e todas as restrições são convexas.

- Método SIMPLEX[15]: Quando a função objetivo e as restrições são funções afins.

- Método quasi-Newton[16]: Quando a função objetivo é diferenciável, unimodal e sem restrições.

- Métodos de programação quadrática sequencial[16]: Quando a função objetivo é diferenciável, unimodal e com restrições.

Os métodos que conseguem, garantidamente, encontrar a solução ótima, se baseiam em premissas fortes a respeito da função-objetivo(diferenciabilidade, convexidade, unimodalidade). Contudo, na prática, os problemas reais não satisfazem tais condições específicas, na verdade, ocorrem funções com estruturas bastante arbitrárias, caóticas e com muitos ótimos locais. Para essas funções, os métodos clássicos de otimização podem convergir para um ótimo local ou não convergir, pois esse tipo de método não possui uma forma que seja capaz de orientá-lo a escapar de tantos ótimos locais até atingir o ótimo global. A necessidade de resolver problemas que possuem funções-objetivo com estruturas arbitrárias tem sido o principal motivo para o desenvolvimento de técnicas de otimização baseadas em heurísticas.

2.1.2 Heurísticas

Uma heurística é uma técnica que busca soluções "razoáveis" sem a garantia matemática de encontrar a solução ótima. Em vez disso, o objetivo é encontrar uma solução melhor do que as alternativas disponíveis. As heurísticas podem encontrar uma solução em questão de minutos, enquanto métodos tradicionais podem levar bilhões de anos ou não encontrar. O papel da heurística é guiar a busca para explorar áreas promissoras e escapar de ótimos locais. Porém, ao longo do processo, a busca vai se concentrando nas melhores regiões na tentativa de se aproximar da solução ótima.

Nos últimos anos, houve um aumento na criação de heurísticas baseadas nos mecanismos de evolução e adaptação dos seres vivos[17][18]. A justificativa para essa abordagem é que esses mecanismos de adaptação conseguem produzir boas respostas para problemas complexos. De acordo com a teoria da evolução de Darwin, a evolução ocorre através da seleção natural e da capacidade dos seres vivos em se adaptar ao ambiente, pois indivíduos mais adequados ao ambiente têm maiores chances de sobreviver e se reproduzir. Além disso, a natureza desenvolveu mecanismos que possibilitam a diversidade, ou seja, a existência de seres diferentes e com características específicas que podem torná-los ainda mais aptos. Este processo de evolução pode ser visto como uma otimização, onde os seres vivos representam as soluções e o meio ambiente representa a função objetivo. Com base nisso, em 1973, Rechenberg[20] iniciou uma classe de algoritmos de otimização denominada Estratégias Evolutivas (EEs).

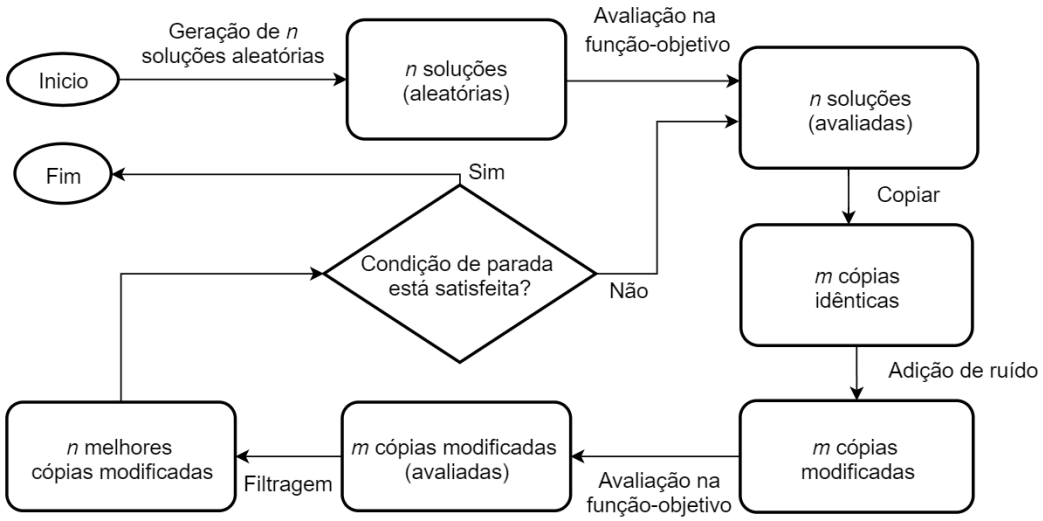
2.1.3 Definição e descrição das EEs

As EEs são algoritmos iterativos que buscam pelo ótimo global utilizando um conjunto de possíveis soluções e aprimorando-as através de pequenas modificações.

Inicialmente, cria-se um conjunto composto por n soluções geradas de forma aleatória e avalia-se a qualidade destas soluções utilizando a função-objetivo. Em seguida, cria-se um novo conjunto composto por m cópias (idênticas) das n soluções geradas inicialmente, com $m > n$. Ruídos aleatórios são adicionados em todas as variáveis de decisão de todas as m cópias, com o objetivo de modificá-las e, possivelmente, melhorá-las. Em seguida, avalia-se a qualidade das m novas soluções e realiza-se uma filtragem. A filtragem tem o objetivo de descartar as piores soluções e manter no novo conjunto apenas as n melhores soluções dentre as m presentes. Ao final, a condição de parada é verificada. Se estiver satisfeita, a iteração do algoritmo é interrompida. Caso contrário, o algoritmo retorna para a etapa de clonagem com as n soluções (que avançaram pela etapa de filtragem) sendo utilizadas como base para as novas cópias.

A figura 2.2 exibe um fluxograma com as etapas do algoritmo.

Figura 2.2: Fluxograma das Estratégias Evolutivas



Fonte: O autor

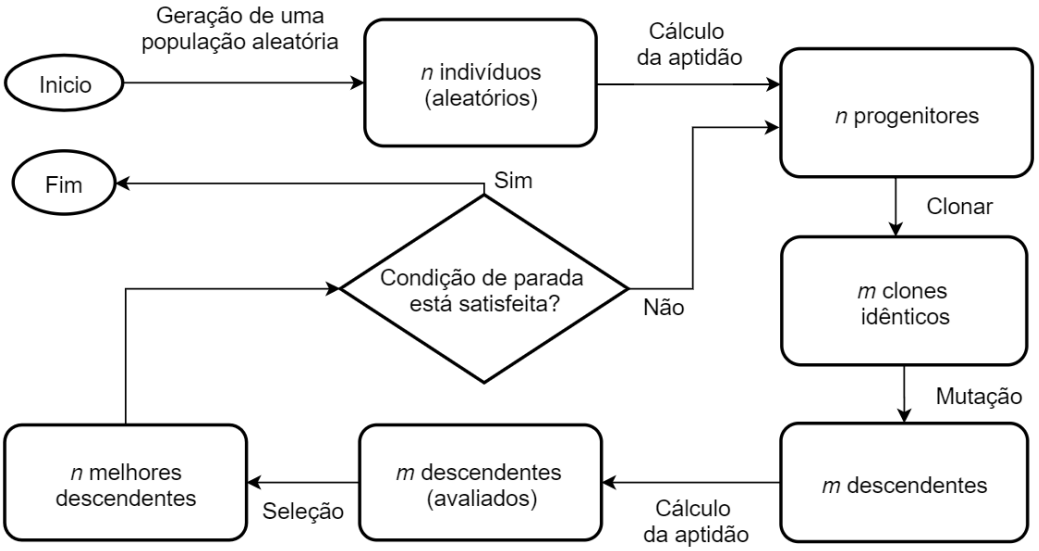
Os elementos de uma Estratégia Evolutiva possuem uma nomenclatura própria e paralela ao ramo da biologia:

- Gene: uma variável de decisão.
- Indivíduo: uma possível solução para o problema.

- População: um conjunto de possíveis soluções para o problema.
- Aptidão(ou *fitness*): o valor da função-objetivo em um determinado ponto(indivíduo).
- Geração: uma iteração completa do algoritmo.
- Mutação: uma modificação realizada no valor de alguma variável de decisão.
- Seleção: o descarte das piores soluções do conjunto de possíveis soluções.
- Progenitor: uma solução utilizada como base para a criação de uma outra solução.
- Descendente: uma solução criada a partir de outra solução(progenitor).

A figura 2.3 exibe o mesmo fluxograma da fig 2.2, porém utilizando a nomenclatura.

Figura 2.3: Fluxograma das Estratégias Evolutivas (com notação)



Fonte: O autor

Esse tipo de algoritmo possui algumas características relevantes:

- Inicia a busca utilizando uma população com indivíduos gerados de forma aleatória (aumentando a diversidade da população inicial).
- Utiliza várias possíveis soluções ao longo da execução, ao contrário dos métodos tradicionais que utilizam apenas uma.
- Não necessita de informações sobre as derivadas da função-objetivo.

- Não necessita que a função-objetivo tenha características específicas.
- Consegue encontrar vários ótimos locais em uma única execução.

Essas características fazem das EEs uma forma eficiente de tratar problemas reais, onde as funções possuem estruturas arbitrárias e muitos ótimos locais.

2.1.4 Clonagem

A clonagem consiste em criar um novo conjunto de indivíduos composto por clones idênticos aos progenitores. Cada progenitor será representado pela mesma quantidade de clones neste novo conjunto. Considerando um exemplo onde a quantidade de progenitores seja 10 e a quantidade de descendentes seja 50, cada progenitor será clonado 5 vezes.

2.1.5 Mutação

A etapa de mutação consiste na adição de pequenos valores aleatórios ao vetor de genes G de um indivíduo. O objetivo desta adição é gerar um novo indivíduo(descendente) que possui a chance de ser melhor(de acordo com a função-objetivo) do que o indivíduo anterior(progenitor). Esse pequeno valor é denominado "ruído", e pode ser positivo ou negativo. Cada um dos genes do indivíduo recebe sua própria quantidade de ruído. Sendo assim, o ruído é um vetor R, onde o elemento R_i representa o valor aleatório que será adicionado ao gene G_i de um determinado indivíduo. Os valores aleatórios do vetor R são gerados de forma que pequenas quantidades ocorram com mais frequência do que grandes quantidades. Para isso, eles são gerados de acordo com uma distribuição Normal, com média nula e variância σ^2. Ou seja, $R_i \sim N(0, \sigma^2)$. Dessa forma, a mutação final é dada por:

$$G' = G + R$$

onde G' representa o vetor de genes pós mutação, G representa o vetor de genes pré mutação, e R representa o vetor de ruídos aleatórios.

2.1.6 Controle do tamanho do passo

O desvio padrão σ utilizado na etapa de mutação determina a intensidade da mutação, que por sua vez determina o tamanho do passo em direção ao ótimo global.

Valores muito altos podem impedir a convergência e valores muito baixos podem tornar a convergência desnecessariamente lenta. Por isso, a escolha deste valor deve ser realizada com cautela. Além disso, Schwefel[21] propôs que o desvio padrão σ também faça parte do processo evolutivo, para que ele seja automaticamente ajustado ao longo da busca. Esse processo é denominado "Auto-adaptação". Na prática, o valor do desvio padrão também será um gene no vetor de genes dos indivíduos, modificando a estrutura dos indivíduos para o seguinte formato:

$$(x_1, x_2, x_3, ..., x_n, \sigma)$$

onde $x_1, x_2, x_3, ..., x_n$ são as variáveis de decisão do problema e σ o valor do desvio padrão. Como nesta abordagem o desvio padrão também faz parte do processo evolutivo, ele também estará sujeito ao processo de clonagem e mutação. Schwefel[21] propõe que a mutação do desvio padrão seja realizada da seguinte maneira:

$$\sigma' = \sigma e^z$$

sendo σ' o valor do desvio padrão pós mutação, σ o valor do desvio padrão pré mutação, e o número de Euler e $z \sim N(0, a^2)$. O valor a é definido como $\frac{1}{\sqrt{n}}$, onde n é a quantidade de genes do indivíduo (excluindo o próprio desvio padrão). Essa forma de mutação do desvio padrão é denominada Adaptação Isotrópica.

2.1.7 Seleção

A etapa de seleção consiste em selecionar os indivíduos que irão "sobreviver" e que, em seguida, serão utilizados como progenitores do próximo conjunto de descendentes. Essa seleção é feita ordenando os indivíduos de acordo com a aptidão e selecionando os n melhores para sobreviverem. Essa forma de seleção é denominada "seleção por truncamento" e é considerada determinística, pois não contém fatores aleatórios.

Inicialmente, a etapa de seleção era aplicada apenas ao conjunto de descendentes, mas Schwefel[21] propôs uma segunda maneira de selecionar os progenitores da próxima geração. Nesta segunda forma, os progenitores da geração atual também são considerados na etapa de seleção. Sendo assim, o conjunto com os "n" progenitores será unido ao conjunto com os "m" descendentes e só então será realizada a ordenação e o truncamento.

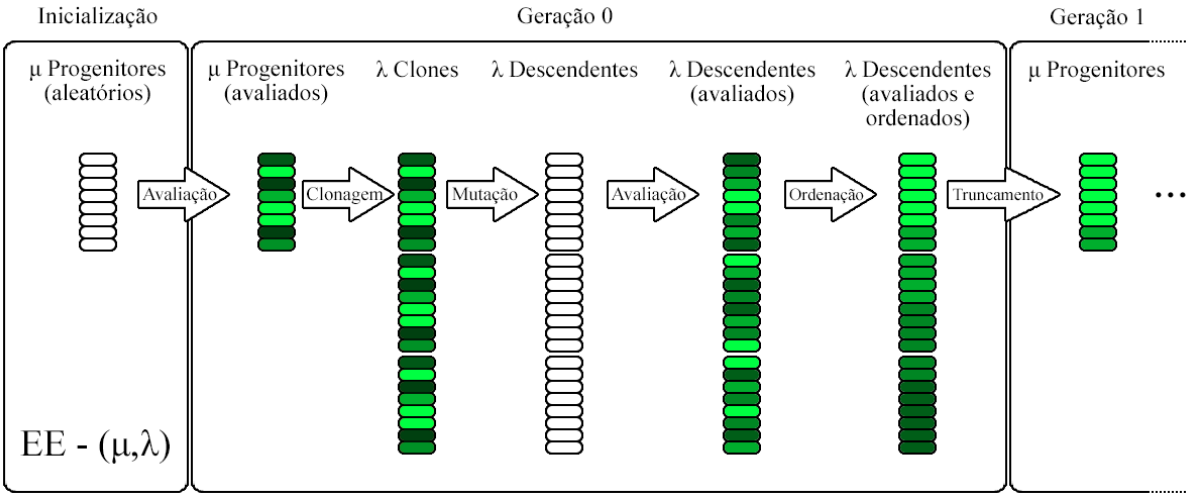
Para diferenciar as EEs quanto ao tipo de seleção, foi estabelecida uma nomenclatura:

- EE-(μ,λ): são as Estratégias Evolutivas que aplicam a seleção apenas no conjunto de descendentes.
- EE-(μ+λ): são as Estratégias Evolutivas que aplicam a seleção no conjunto união entre o conjunto de progenitores e o conjunto de descendentes.

Esta nomenclatura classifica as EEs de acordo com a quantidade de progenitores, a quantidade de descendentes e o tipo de seleção executada. A letra grega "μ" (mi) representa a quantidade de progenitores. A letra "λ" (lambda) representa a quantidade de descendentes que serão gerados e os símbolos "+" e "," representam os dois tipos de seleção. Estas duas versões de EEs diferem apenas quanto à composição do conjunto de indivíduos que avançará para a etapa de seleção.

As figuras 2.4 e 2.5 exibem esquemas que explicitam as diferenças entre os dois métodos. Cada bloco retangular colorido representa um indivíduo. A cor branca representa uma aptidão desconhecida. A cor verde claro representa uma aptidão alta e a cor verde escuro representa uma aptidão baixa.

Figura 2.4: Esquema EE-(μ,λ)

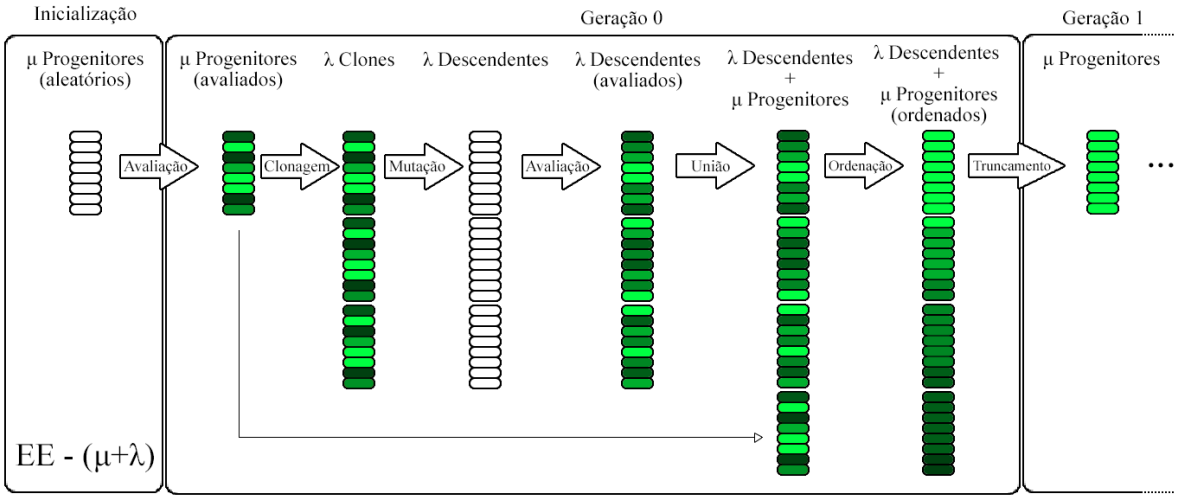


Fonte: O autor

Pela figura 2.5, nota-se que o conjunto de progenitores é agregado ao conjunto de descendentes para formar o conjunto de candidatos a próximos progenitores. Essa

estratégia possibilita que um indivíduo tenha um "ciclo de vida" maior que apenas uma geração.

Figura 2.5: Esquema EE-(μ+λ)



Fonte: O autor

2.1.8 Condição de Parada

A condição de parada consiste em elaborar uma condição que interrompa a execução do algoritmo, caso contrário, a busca nunca chegaria ao fim. Diversas condições de parada podem ser utilizadas, tais como:

- A obtenção de uma solução com aptidão considerada satisfatória;
- Tempo máximo de execução;
- Quantidade máxima de gerações;
- Quantidade máxima de gerações sem melhoria.

2.1.9 Pseudocódigo

O Algoritmo 1 demonstra o funcionamento das Estratégias Evolutivas descritas nas seções anteriores.

Algoritmo 1 Estratégias Evolutivas $(\mu \overset{+}{,} \lambda)$

1: **Requer:** μ, λ, σ, $tipoEE$
2: **Progenitores** $\leftarrow \{(\mathbf{x}_k \leftarrow aleatorio(), \sigma, f(\mathbf{x}_k)) : k = 1, ..., \mu\}$
3: **enquanto** $condicaoParada$ **for** $falsa$ **faça**
4: **Clones** $\leftarrow clonar($**Progenitores**$, \lambda)$
5: **Descendentes** $\leftarrow mutar($**Clones**$)$
6: $avaliar($**Descendentes**$)$
7: **se** $tipoEE$ **for** $(\mu + \lambda)$ **então**
8: **Descendentes** $\leftarrow$ **Descendentes** $\cup$ **Progenitores**
9: **fim se**
10: $ordenar($**Descendentes**$)$
11: **Progenitores** $\leftarrow truncar($**Descendentes**$, \mu)$
12: **fim enquanto**
13: **retorne** $melhor($**Progenitores**$)$

2.2 Redes Neurais Artificiais

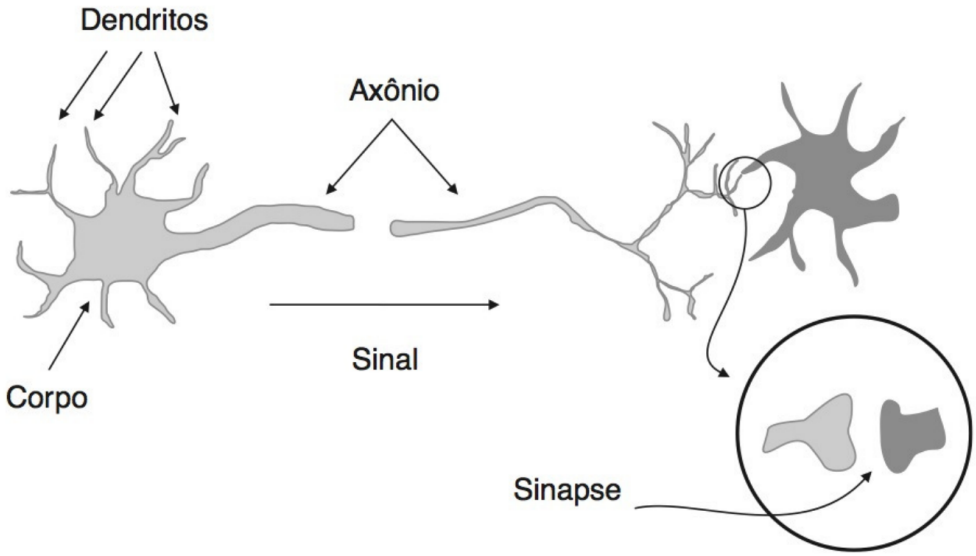
As informações presentes nesta seção são referentes ao livro "*Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina*" de Faceli et al[22].

A busca por modelos computacionais e matemáticos do cérebro humano começou na década de 1940, na mesma época em que os primeiros computadores eletrônicos estavam sendo desenvolvidos. Naquela época, um dos primeiros algoritmos propostos foram as Redes Neurais Artificiais (RNA). As RNA são modelos matemáticos que se baseiam no comportamento do sistema nervoso dos seres humanos, mais especificamente, no funcionamento do cérebro humano. A função delas é imitar a habilidade de aprender e adquirir conhecimento que o cérebro possui. No entanto, para compreender melhor o seu funcionamento, se faz necessário compreender o sistema nervoso humano.

2.2.1 Sistema Nervoso Humano

O sistema nervoso é um conjunto de células responsável por controlar o comportamento dos seres vivos. O elemento fundamental do sistema nervoso é o neurônio, que possui uma característica denominada excitabilidade. A excitabilidade permite que o neurônio responda a estímulos e transmita impulsos nervosos a outros neurônios ou a outras células do corpo. O neurônio é composto, basicamente, por dendritos, axônio e o corpo celular. A figura 2.6 exibe uma ilustração de um neurônio biológico simplificado.

Figura 2.6: Estrutura simplificada de um neurônio biológico



Fonte: [22]

Os dendritos são as regiões do neurônio especializadas em receber estímulos nervosos de outros neurônios ou do ambiente e encaminhar para o corpo celular. O corpo celular é responsável por combinar os impulsos nervosos recebidos por todos os dendritos do neurônio e os processar. Dependendo da intensidade e da frequência dos impulsos recebidos, o corpo celular pode gerar um novo impulso e o enviar para o axônio. Os axônios, por sua vez, são regiões do neurônio especializadas em conduzir os impulsos nervosos gerados pelo corpo celular até os dendritos de outros neurônios. O axônio de um humano adulto pode atingir comprimentos superiores a um metro. O local de contato entre o axônio de um neurônio e o dendrito de outro é denominado "sinapse". As sinapses são os elementos que fazem o intermédio entre as comunicações de dois neurônios e podem ser excitatórias ou inibitórias.

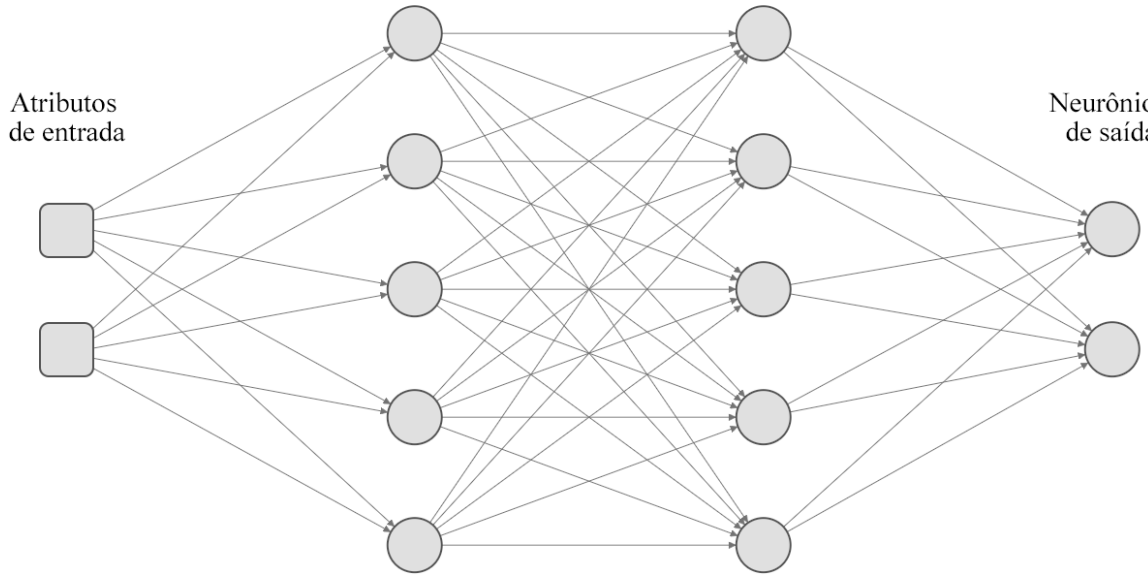
O cérebro humano possui por volta de 10 a 500 bilhões de neurônios. Segundo estimativas, esses neurônios estão agrupados em aproximadamente 1000 módulos, cada um contendo aproximadamente 500 redes neurais. Além disso, cada neurônio pode estar conectado a milhares de outros neurônios, formando uma rede densa e altamente paralela. Apesar dos neurônios possuírem um tempo de resposta na ordem dos 0,001 segundo(muito mais lento que um processador digital), o paralelismo do cérebro humano torna o processamento das informações muito eficiente, permitindo que ele realize diversas tarefas complexas mais rapidamente que um computador digital.

2.2.2 Definições e descrições das RNA

De acordo com Faceli[22]: "As RNAs são sistemas computacionais distribuídos compostos de unidades de processamento simples, densamente interconectadas." Essas unidades de processamento são denominadas "neurônios artificiais". Os neurônios artificiais são agrupados em camadas e se comunicam uns com os outros através de conexões unidirecionais. Cada uma dessas conexões possui um peso associado, que serve para aumentar ou diminuir a magnitude da informação recebida por esta conexão. Os valores dos pesos são os responsáveis por armazenar e codificar o conhecimento obtido pela rede neural e são ajustados em um processo denominado "aprendizado" ou "treinamento". O ajuste dos pesos tem por objetivo regular o comportamento do modelo para que ele responda da maneira desejada. A figura 2.7 exibe o esquema de uma rede neural com várias camadas de neurônios artificiais. Essa rede recebe dois valores como entrada e produz

dois valores como saída.

Figura 2.7: Rede neural artificial



Fonte: O autor.

Neurônio artificial

O neurônio artificial é o principal elemento de uma rede neural, é a unidade fundamental de processamento de informações. De forma análoga aos neurônios do cérebro humano, os neurônios artificiais possuem terminais de entrada que recebem informações (valores). Esses valores são ponderados (utilizando o peso associado à respectiva conexão) e somados. A entrada E recebida pelo neurônio é definida por:

$$E = \sum_{i=1}^{c} x_i p_i$$

onde x_i é o valor recebido pela conexão i, p_i é o peso associado a esta conexão e c é a quantidade total de conexões que este neurônio possui.

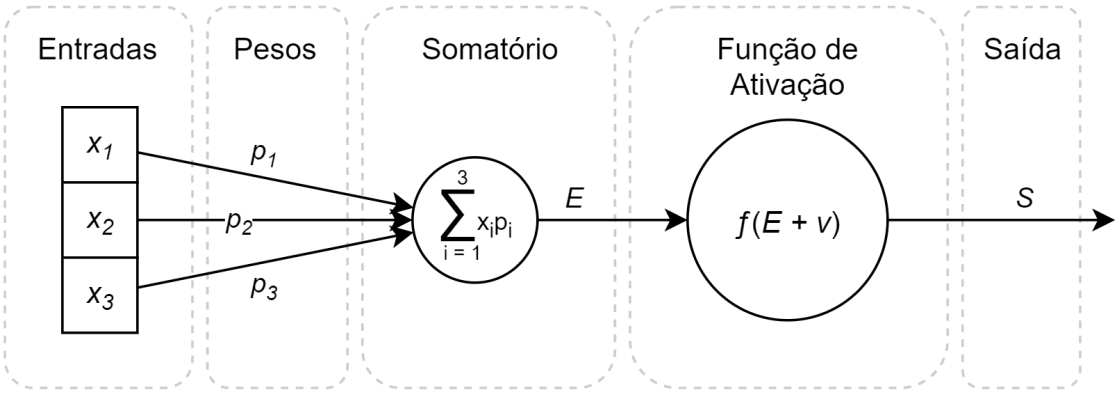
A função do peso p_i é controlar a intensidade da contribuição que aquela conexão possui na ativação final do neurônio. Se p_i for um valor muito grande, x_i será multiplicado por um número muito grande e irá aumentar sua magnitude, tornando maior sua contribuição para a soma final. Os pesos podem assumir valores positivos (conexão excitatória e contribuição positiva), valores negativos (conexão inibitória e contribuição negativa) ou zero (conexão ausente e nenhuma contribuição).

Após computada a entrada E, um termo v (denominado "viés") é adicionado a este valor. Em seguida, o valor $E + v$ é passado como argumento para uma função matemática que realizará o processamento da informação, simulando o comportamento do corpo celular. Dessa forma, a saída S do neurônio é definida por:

$$S = f(E + v) = f((\sum_{i=1}^{c} x_i p_i) + v)$$

onde f é uma função matemática denominada "função de ativação" e v é o valor do viés. O papel do viés v é representar o quão "difícil" é a ativação do neurônio. Se o viés for um valor que possui um módulo muito grande mas com sinal negativo, ele irá contribuir muito negativamente para a entrada final $E + v$, dificultando a ativação do neurônio. Pois será necessário que os valores de entrada x_i tenham grandes magnitudes para que $E + v$ se torne positivo. Da mesma forma, se v possui um valor com módulo muito grande mas com sinal positivo, a ativação do neurônio se torna fácil, pois qualquer entrada (até mesmo um zero) é capaz de tornar $E + v$ positivo. A figura 2.8 exibe um modelo simples de um neurônio artificial.

Figura 2.8: Neurônio artificial



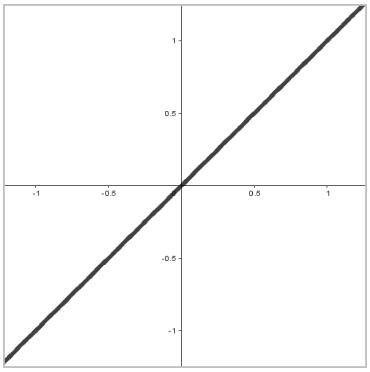
Fonte: O autor.

Funções de ativação

Diversas funções matemáticas foram propostas na literatura para serem usadas como funções de ativação em redes neurais. A figura 2.9 exibe a equação e o gráfico de algumas das mais populares.

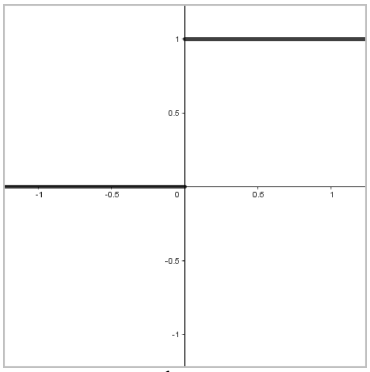
Figura 2.9: Funções de ativação (populares)

(a) Linear



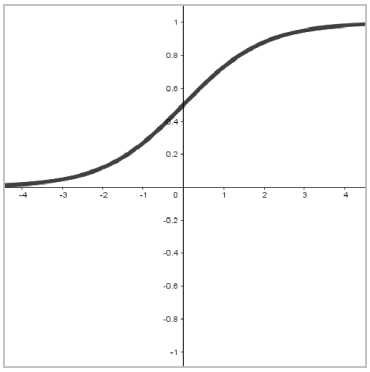
$f(x) = x$

(b) Limiar ($L = 0$)



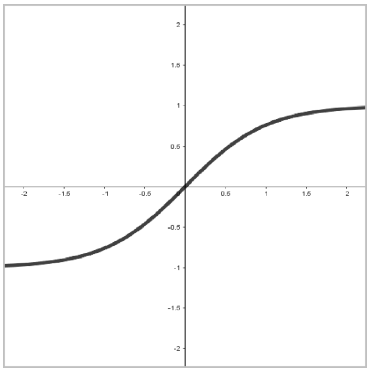
$$f(x) = \begin{cases} 0, & x \leq L \\ 1, & x > L \end{cases}$$

(c) Sigmoidal



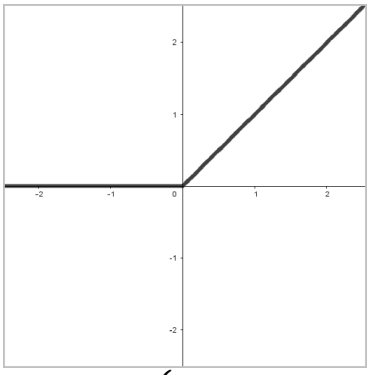
$f(x) = \frac{1}{1+e^{-x}}$

(d) Tangente Hiperbólica



$f(x) = tanh(x)$

(e) ReLU



$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

Fonte: O autor.

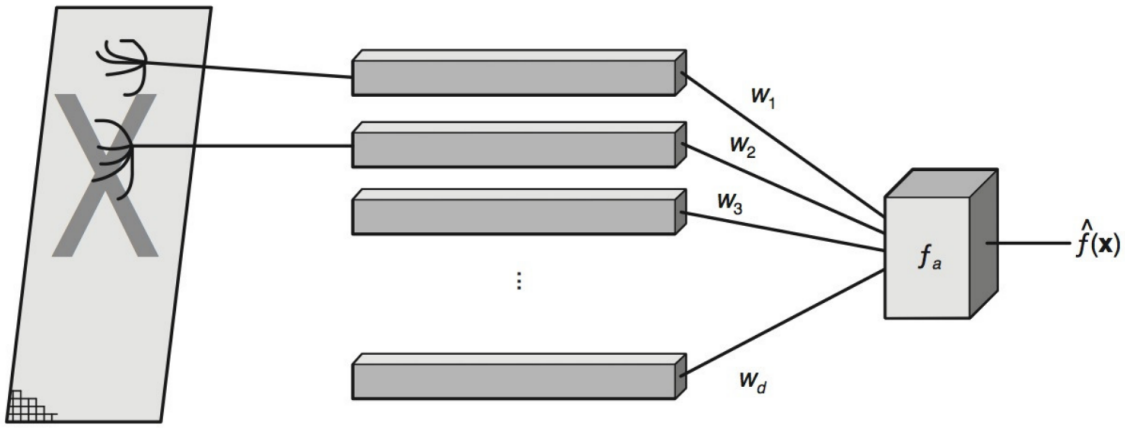
- Função Linear: retorna como saída o próprio valor da entrada;
- Função Limiar: retorna 1 se a entrada for maior que o limite definido, e 0 caso contrário.
- Função Sigmoidal: é uma versão contínua e diferenciável da função limiar;
- Função Tangente Hiperbólica: similar a função sigmoidal, porém com uma amplitude maior, sendo capaz de produzir valores entre -1 e 1;
- Função *Rectified Linear Unit* (ReLU): uma combinação entre a função linear e a limiar.

Normalmente, o neurônio é considerado ativado quando produz uma saída com um valor maior que zero. Porém, a função sigmoidal é uma exceção a esta convenção. Pois sua saída pode assumir quaisquer valores entre 0 e 1 mas apenas os valores maiores que 0,5 são considerados como "neurônio ativado".

Perceptron

Em 1958, Rosenblatt[23] implementou a primeira rede neural, denominada "rede perceptron". A rede perceptron possuía uma máscara que recebia os objetos de entrada e processava essa entrada usando seu único neurônio. O neurônio da perceptron utilizava o modelo proposto por McCulloch-Pitts[24]. Neste modelo os neurônios executam funções lógicas simples (como por exemplo, a função limiar), e por isso eram denominados "Unidades Lógicas com Limiar" (LTU, do inglês *Logic Threshold Unit*). Apesar de possuir apenas um neurônio artificial, a perceptron demonstrou uma acurácia significativa em vários problemas de classificação. A figura 2.10 exemplifica a estrutura de uma perceptron.

Figura 2.10: Estrutura de uma perceptron



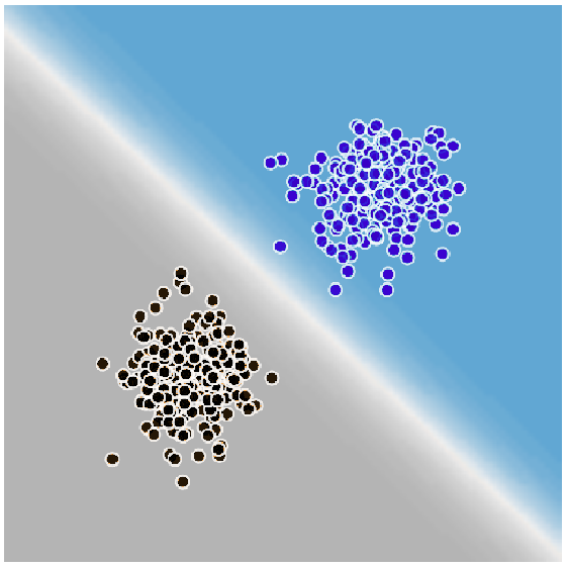
Fonte: [22].

Rosenblatt[23] também provou o teorema de convergência da rede perceptron, que afirma que, se um conjunto de entradas for linearmente separável, a perceptron conseguirá realizar a classificação. Para verificar se duas classes são linearmente separáveis, basta plotar os atributos dos seus objetos em um plano (ou em um espaço n-dimensional), e verificar se existe uma reta (ou um hiperplano) que separa os objetos destas classes. A figura 2.11a demonstra essa situação. No entanto, a perceptron possui uma limitação na sua capacidade de classificação e não consegue classificar conjuntos de objetos que não

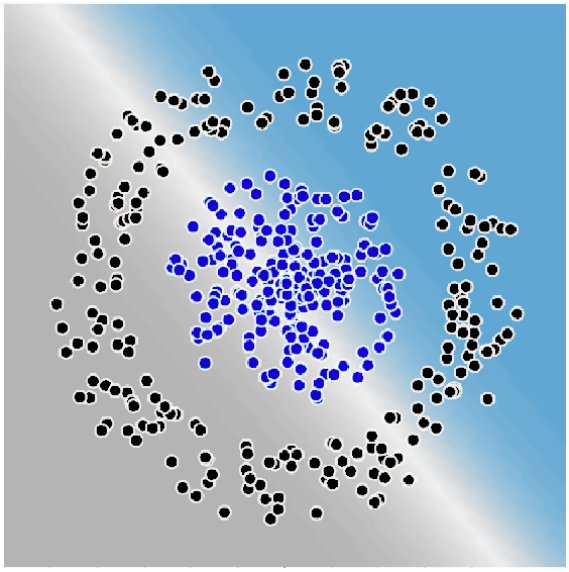
sejam linearmente separáveis. A figura 2.11b mostra dois conjuntos que uma perceptron não consegue classificar.

Figura 2.11: Exemplos de conjuntos de objetos

(a) Conjuntos linearmente separáveis



(b) Conjuntos não-linearmente separáveis



Fonte: O autor.

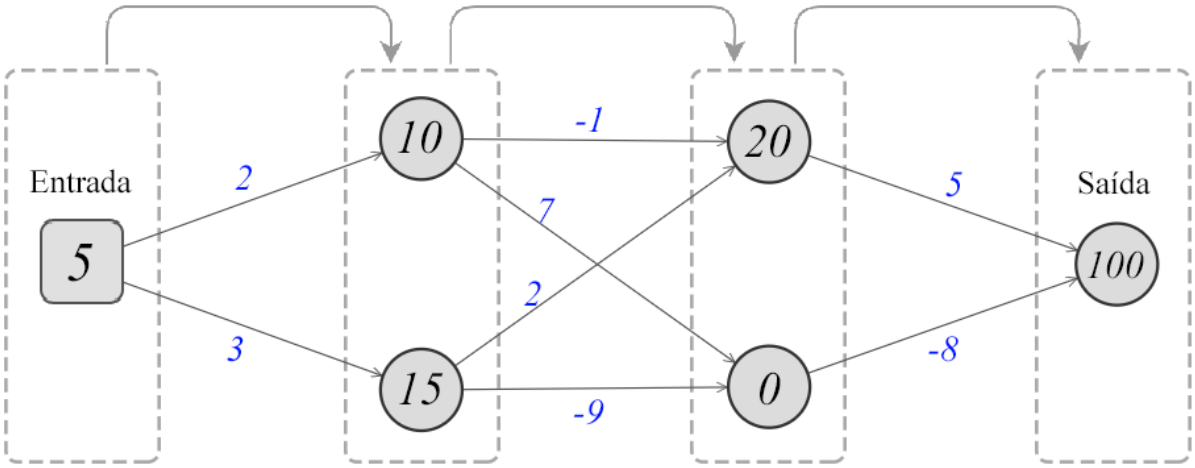
Para resolver este problema, a solução mais utilizada é adicionar camadas intermediárias na rede. Pois, de acordo com Cybenko[25], uma rede neural com apenas uma camada intermediária e neurônios suficientes é capaz de aproximar qualquer função contínua. Vale ressaltar que essa afirmação é válida apenas para redes que utilizam funções de ativação não-linear. Pois redes com mais de uma camada que utilizam funções de ativação linear são equivalentes a redes com uma única camada.

2.2.3 Redes Neurais Multicamadas

Quando uma rede neural possui mais de uma camada, os neurônios recebem em seus terminais de entrada as saídas dos neurônios da camada anterior, e enviam suas saídas para os terminais dos neurônios da próxima camada. Uma rede desse tipo é denominada “rede multicamadas”. A última camada é denominada “camada de saída” e as restantes são denominadas “camadas ocultas” ou “intermediárias”. A camada de saída é a responsável por calcular a resposta final da rede para a entrada fornecida, enquanto as camadas intermediárias são responsáveis pelo reconhecimento de padrões e pela lógica do processamento da informação. O processamento das informações de entrada é realizado

da primeira camada intermediária até a camada de saída, camada por camada, sequencialmente. Esse processo é denominado *feedforward* e está ilustrado na figura 2.12. A figura 2.12 exibe o processo de *feedforward* sendo realizado em uma rede que utiliza a função ReLU como função de ativação em todos os neurônios e possui todos os vieses iguais a zero.

Figura 2.12: O processo de *feedforward*.



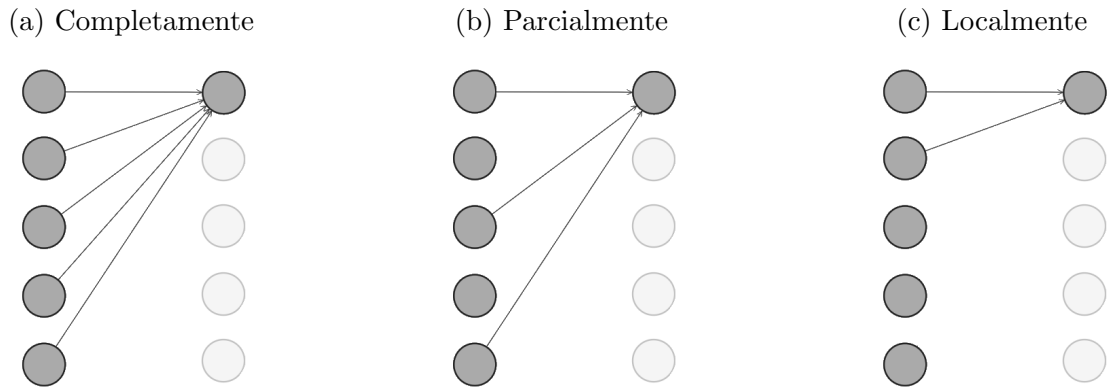
Fonte: O autor.

Em uma RNA multicamadas os neurônios podem se conectar de várias formas diferentes, e por isso, são divididas nas seguintes categorias:

- Completamente conectada: Quando todos os neurônios da rede se conectam com todos os neurônios da camada anterior a sua.
- Parcialmente conectada: Quando os neurônios estão conectados a alguns(mas não todos) neurônios da camada anterior a sua.
- Localmente conectada: Quando os neurônios estão conectados a alguns neurônios da camada anterior a sua e esses neurônios estão, de certa forma, próximos uns dos outros.

A figura 2.13 ilustra estas três formas de conexão.

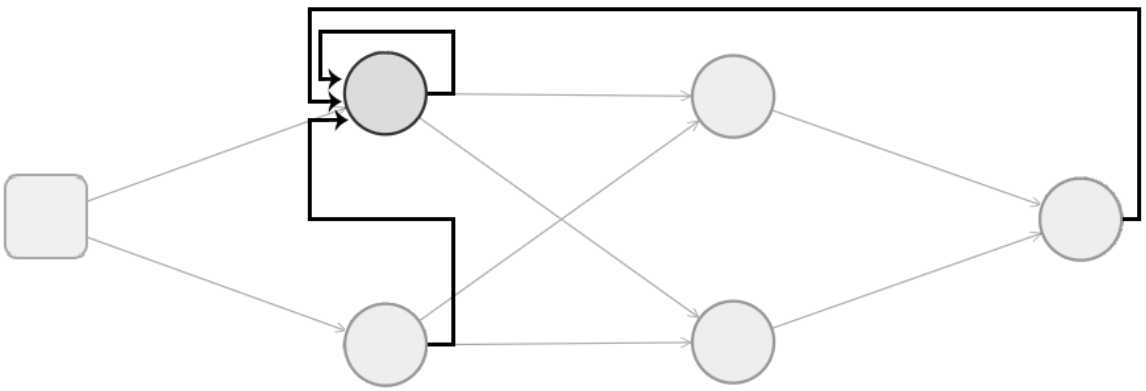
Figura 2.13: Formas de conexão entre neurônios



Fonte: O autor.

Além das várias formas que os neurônios de uma camada possuem de se conectar aos neurônios da camada anterior, eles também podem apresentar conexões de retroalimentação. Isto é, o neurônio pode receber no seu terminal de entrada a saída dos neurônios que estão posicionados na mesma camada ou em camadas posteriores à dele. Em conexões do tipo retroalimentação, o valor que um neurônio recebe como entrada é a saída do outro neurônio produzida no *feedforward* anterior. A figura 2.14 ilustra várias formas de retroalimentação. Note que o neurônio destacado também recebe em sua entrada a saída que ele mesmo produziu no processamento (*feedforward*) anterior.

Figura 2.14: Conexões de retroalimentação



Fonte: O autor.

As redes neurais que possuem retroalimentação são denominadas "redes neurais recorrentes" e são recomendadas para problemas que possuem a necessidade de processar informações sequenciais ou simular sistemas dinâmicos. Exemplos desses problemas são: o processamento de linguagem natural e o controle de braços robóticos. As redes neurais

que não possuem retroalimentação são denominadas “redes neurais *feedforward*”.

Dito isto, a topologia de uma rede neural artificial é definida pelos seguintes fatores:

- A quantidade de camadas que a rede possui.
- A quantidade de neurônios em cada camada.
- O modo de conexão (completamente, localmente ou parcialmente conectada.)
- A presença (ou não) de conexões com retroalimentação.

Perceptron Multicamadas

Quando uma perceptron possui uma ou mais camadas intermediárias, ela é denominada “Perceptron Multicamadas” (MLP, do inglês *multilayer perceptron*). Em uma MLP, cada neurônio define uma função específica, que por sua vez, é uma combinação das funções dos neurônios da camada anterior à dele. Conforme o processamento avança de uma camada para a próxima, a função definida por cada neurônio se torna mais complexa. A figura 2.15 exibe este processo de complexificação.

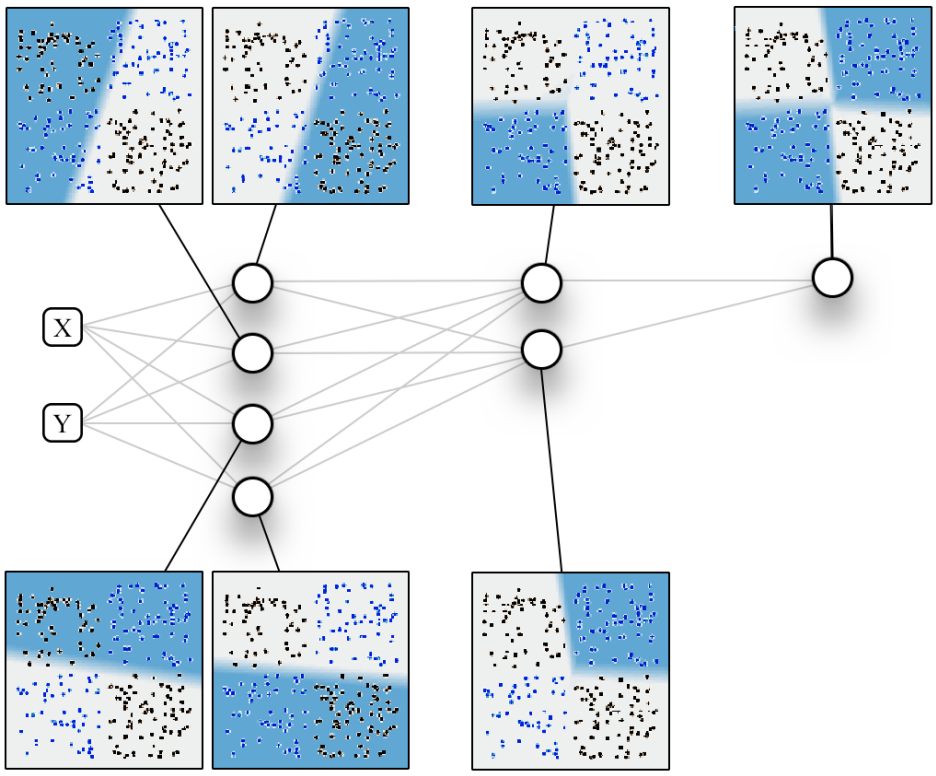
Na figura 2.15, os neurônios da primeira camada definem planos (ou hiperplanos). Na segunda camada, os neurônios combinam as saídas dos neurônios da primeira camada para conseguir definir regiões convexas. Na camada de saída, o neurônio utiliza as regiões convexas para definir regiões com formatos arbitrários. A combinação das funções de cada neurônio da RNA determina a função final que a MLP representará.

Em problemas de classificação, cada neurônio da camada de saída está associado a uma classe distinta. Por isso, ao final do *feedforward*, a classe associada ao neurônio com a maior saída será considerada a classe a qual a entrada pertence.

2.2.4 Treinamento e teste

O treinamento é o processo de ajustar os valores dos pesos e dos vieses da rede neural com o objetivo de minimizar os erros cometidos por ela na classificação dos dados. Por outro lado, o teste é o processo de expor a rede a dados inéditos (ou seja, dados que ela nunca “viu”, dados que não foram usados na etapa de treinamento) com o objetivo de verificar a generalização da rede e a sua acurácia na classificação de dados com rótulos desconhecidos.

Figura 2.15: Funções definidas pelos neurônios de uma perceptron.



Fonte: O autor

Muitos algoritmos foram propostos na literatura para realizar o ajuste dos pesos de uma RNA. Eles consistem em uma sequência de passos/regras que definem quando e de que forma o valor do peso deve ser alterado. Dentre esses algoritmos, existem três paradigmas que se destacam:

- Aprendizado supervisionado: Neste tipo de aprendizado, os dados são apresentados para a rede e a saída produzida por ela é armazenada e comparada a um "gabarito", que contém os valores corretos que ela deveria produzir. De posse da saída gerada e da saída desejada, o valor do erro é calculado e os pesos da rede são ajustados de forma a reduzir este valor. O algoritmo mais popular para o aprendizado supervisionado é o *backpropagation*, proposto por Rumelhart[26], em 1986.

- Aprendizado não supervisionado: Ao contrário do aprendizado supervisionado, no aprendizado não supervisionado os rótulos dos dados não são fornecidos para a rede neural. A rede deve encontrar a função que classifica os dados utilizando outras estratégias. Uma das estratégias mais antigas e populares é a Regra de Hebb[27]. Essa regra afirma que a força da conexão sináptica entre dois neurônios deve ser

aumentada quando eles são ativados simultaneamente. Na prática, isto significa que o peso da conexão entre eles será aumentado.

- Aprendizado por reforço: Diferentemente do aprendizado supervisionado e do aprendizado não supervisionado, neste tipo de aprendizado, a rede neural é colocada para interagir com um ambiente e receber recompensas ou punições de acordo com suas ações neste ambiente. As recompensas servem para estimular as ações bem sucedidas, enquanto as punições servem para inibir as ações mal-sucedidas. Um dos algoritmos mais populares na área de aprendizado por reforço é o *Q-Learning*[28], que, em conjunto com as redes neurais, forma o *Deep Q-Learning*[29].

Além destes três paradigmas, as meta-heurísticas (grupo no qual estão incluídas as estratégias evolutivas) também possuem a capacidade de ajustar os pesos de uma rede neural e por isso podem ser consideradas uma alternativa viável aos algoritmos tradicionais.

Contudo, alguns procedimentos devem ser realizados antes e durante o treinamento de uma RNA.

Antes do treinamento

Visando impedir que a rede neural fique sobreajustada aos dados de treinamento e perca a sua generalização (o conceito de "sobreajuste" será abordado mais adiante), é recomendado que os dados disponíveis sejam separados em dois conjuntos distintos: conjunto de treinamento e conjunto de validação. O conjunto de treinamento será o conjunto de dados que a rede neural utilizará para ajustar seus pesos. Enquanto o conjunto de validação será o conjunto que servirá para verificar o quão bem generalizada a rede está. Isto é, se a acurácia da classificação para dados desconhecidos está elevada. Geralmente, é utilizada uma proporção de 7:3 ou 8:2 para a cardinalidade dos conjuntos de treino e validação, respectivamente. Isto é, 70% (ou 80%) dos dados devem ser destinados ao conjunto de treinamento e os outros 30% (ou 20%) destinados ao conjunto de validação.

Além da separação destes dois conjuntos de dados, é importante também definir como os pesos e os vieses da rede serão inicializados. Pois, se forem inicializados de forma inadequada, podem atrapalhar o processo de treinamento, tornando-o mais lento ou até mesmo impedindo a convergência. Em 2010, Glorot[30] propôs uma forma de inicializar

os pesos que posteriormente foi denominada “Inicialização Xavier”. Esta abordagem é aplicada em redes neurais que utilizam funções de ativação do tipo sigmoide ou tangente hiperbólica. De acordo com a inicialização Xavier, os vieses são inicializados com o valor zero e os pesos são inicializados utilizando uma distribuição uniforme:

$$P_{ijk} \sim U(-a, a)$$

sendo:

- P_{ijk} o peso da conexão i do neurônio j da camada k;
- $a = \frac{1}{\sqrt{n}}$;
- n a quantidade de entradas do neurônio j;

Alguns anos depois, em 2015, He[31] propôs uma forma de inicializar os pesos das redes neurais que utilizam a função de ativação ReLU. Essa inicialização é denominada “Inicialização He”. De acordo com ela, os vieses também são inicializados com zero e os pesos são inicializados de acordo com uma distribuição normal:

$$P_{ijk} \sim N(0, \sigma^2)$$

onde:

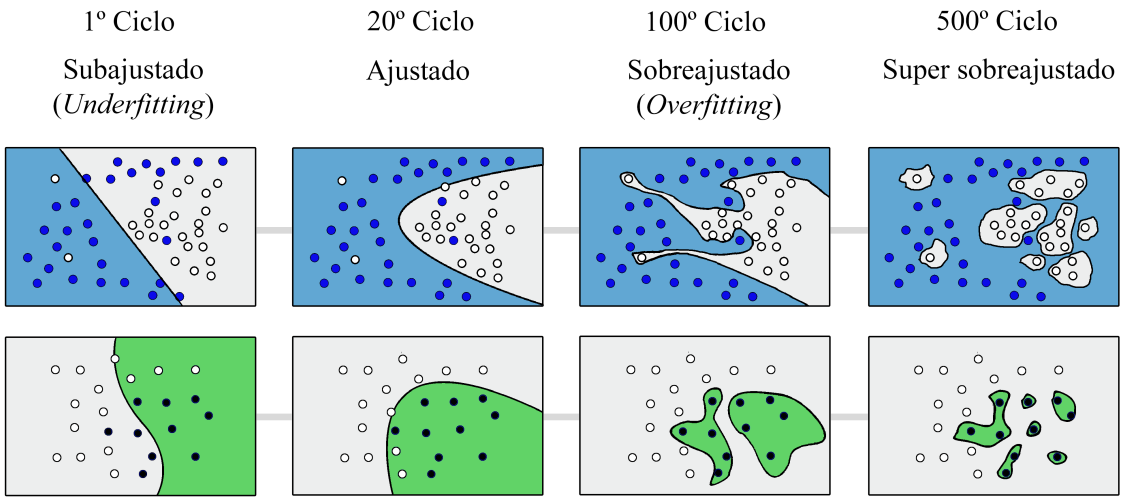
- P_{ijk} é o peso da conexão i do neurônio j da camada k;
- $\sigma = \sqrt{\frac{2}{n}}$;
- n é a quantidade de entradas do neurônio j;

Durante o treinamento

Durante o treinamento, a rede neural irá ajustar os valores dos seus pesos para diminuir os erros cometidos na classificação dos dados de treinamento. Porém, se o treinamento se mantiver por muito tempo (muitos ciclos), a rede pode começar a se “sobreajustar” e aos poucos perder a sua capacidade de generalização. O sobreajuste (em inglês, *overfitting*) acontece quando a rede neural se ajusta tanto aos dados de treinamento que ela deixa de representar um padrão geral e passa a “decorá-los”. A figura 2.16 exibe

dois exemplos de treinamentos onde o sobreajuste ocorreu. No 1º ciclo, a rede ainda não aprendeu nada, pois o treino acabou de começar. No 20º ciclo, a rede está em seu formato ideal. No 100º ciclo, um sobreajuste ocorreu. Áreas que deveriam pertencer a uma classe, estão pertencendo a outra. No 500º ciclo, o processo de sobreajuste foi levado ao extremo, fragmentando completamente as regiões de classificação.

Figura 2.16: O processo de sobreajuste



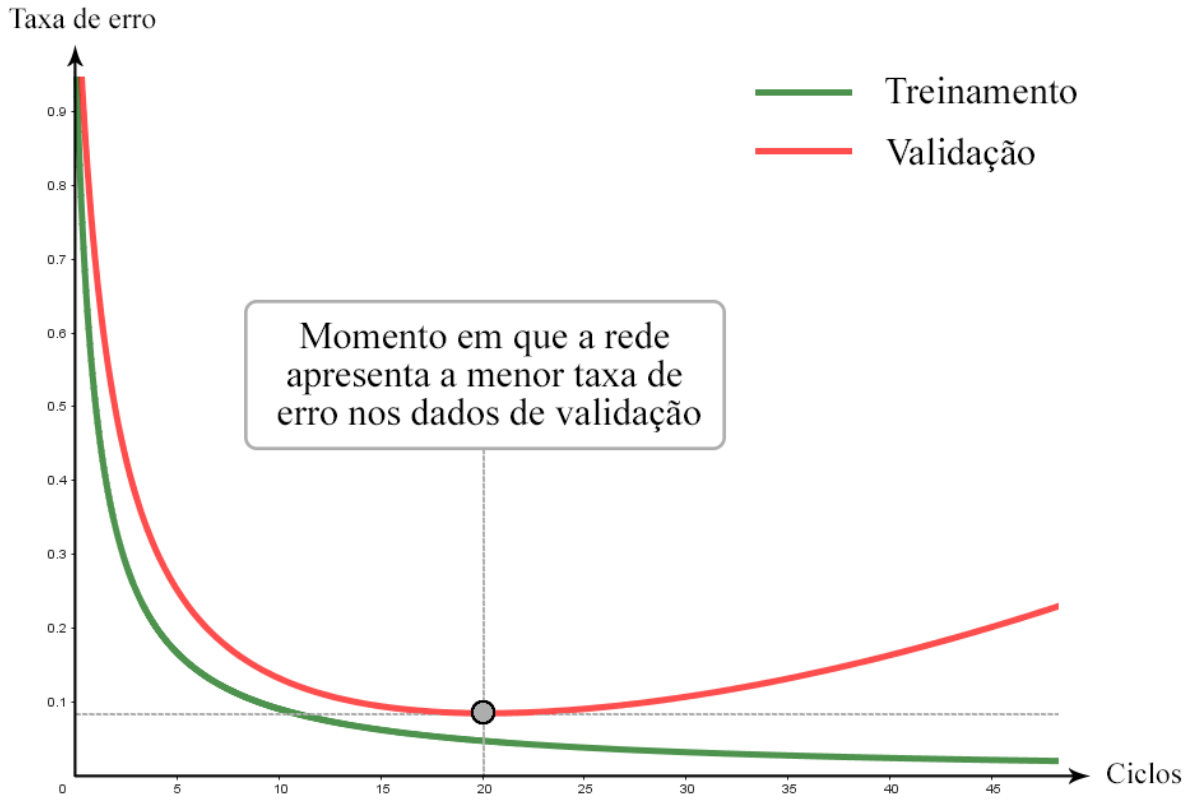
Fonte: O autor

Como o objetivo do treinamento é diminuir o erro cometido pela rede nos dados de treinamento, se o treino não for interrompido no momento correto, a tendência é que a rede comece a decorar os dados, visando atingir uma taxa de erro igual a zero. Uma das formas de se evitar que a rede fique super ajustada é utilizar a validação cruzada com *early stop*.

De acordo com Haykin[33], a validação cruzada consiste em apresentar para a rede neural os dados do conjunto de validação a cada n ciclos de treinamento e calcular a sua acurácia. Caso a taxa de erros cometidos pela rede nos dados de validação comece a aumentar, um super ajuste pode estar acontecendo. Por isso, uma parada prematura (*early stop*) é realizada e o treinamento é finalizado. A figura 2.17 exibe um gráfico que demonstra esse comportamento. Note que no início do treinamento ambas as taxas de erro (treinamento e validação) são elevadas, pois a rede ainda não aprendeu o padrão nos dados. Ao longo do treinamento as duas taxas de erro tendem a cair à medida que os pesos são ajustados. Em certo ponto, a taxa de erro do conjunto de treinamento continua diminuindo enquanto a taxa de erro no conjunto de validação começa a aumentar. Isto é

um indício de que a rede parou de aprender e um sobreajuste está ocorrendo.

Figura 2.17: Gráfico com as taxas de erro por ciclo



Fonte: O autor

2.2.5 Projeto de Arquitetura

Em um projeto envolvendo RNA, é necessário definir a sua arquitetura. Ela engloba a topologia da rede e as funções de ativação de cada neurônio. Escolher a arquitetura mais promissora para um conjunto de dados não é uma tarefa trivial, pois a quantidade adequada de neurônios para as camadas intermediárias dependem de alguns fatores, como por exemplo:

- A quantidade de exemplos (dados) disponíveis para o treinamento.
- A quantidade de ruído (imprecisões) contida nos dados.
- A complexidade da função que se deseja aproximar.
- A distribuição estatística dos dados de treinamento.

Geralmente é realizado um processo de tentativa de erro, onde diferentes arquiteturas são verificadas visando descobrir quais apresentam os melhores resultados. Nesse processo, cada arquitetura verificada é treinada e avaliada de acordo com a acurácia apresentada no treinamento. Embora seja o método mais utilizado para determinar a melhor arquitetura, essa abordagem apresenta um elevado custo de tempo e esforço, pois o treinamento será realizado várias vezes. Algumas heurísticas podem ser aplicadas visando diminuir o tempo de busca pela melhor arquitetura. Uma estratégia comum é iniciar a exploração verificando primeiro as RNAs que contêm apenas uma camada intermediária, pois esse tipo de rede neural já possui uma capacidade de aprendizado considerável.

2.2.6 Prós e Contras

As RNAs têm algumas características que as tornam populares, como por exemplo, a sua capacidade de generalização e a sua tolerância a falhas e ruídos[32][33]. Essas características contribuem para o bom desempenho das RNAs, resultando em baixas taxas de erros em diversas aplicações, principalmente em tarefas de visão computacional e robótica. No entanto, em algumas aplicações, o desempenho não é o único fator relevante, compreender a lógica que a rede neural está usando para tomar suas decisões também é necessário. Por isso, uma crítica comum ao uso de RNAs, é a complexidade em entender como e por que elas tomam suas decisões. Essa dificuldade está relacionada ao fato de que o conhecimento da rede está armazenado em um grande número de parâmetros, que são manipulados por meio de complicadas fórmulas matemáticas. Como resultado, as RNAs frequentemente são consideradas como "caixas-pretas". Além disso, procurar a melhor arquitetura para a rede neural também não é um processo bem conhecido/definido, sendo muitas vezes apelidado de *black art*.

Capítulo 3

Desenvolvimento

O objetivo deste trabalho consiste em avaliar a eficácia do uso das estratégias evolutivas como método de treinamento para redes neurais perceptron no contexto de jogos digitais. Para atingir este objetivo, as estratégias evolutivas, as redes neurais e os jogos selecionados foram completamente implementados. Isto é, não foram utilizadas bibliotecas específicas de EE, RNA ou jogos na implementação destes elementos. No entanto, uma biblioteca gráfica teve de ser utilizada para implementar a parte gráfica dos jogos e realizar a comunicação com os dispositivos de entrada e saída (mouse e teclado). Os detalhes desta biblioteca serão descritos na seção "Jogos".

Para facilitar a interação do autor com o treino e com o teste das redes neurais, e torná-lo também um processo mais visual, uma interface gráfica foi desenvolvida utilizando a mesma biblioteca gráfica aplicada nos jogos. Os detalhes desta interface serão descritos na seção "Interface Gráfica".

Após as implementações, os algoritmos foram configurados e colocados para interagir entre si. Portanto, este capítulo apresentará:

- a forma como os algoritmos (EE e RNA) foram implementados;
- a forma como foram configurados (parâmetros utilizados);
- a forma como os elementos (EE, RNA e jogos) interagem entre si;
- o funcionamento de cada jogo implementado;
- o funcionamento da interface gráfica desenvolvida.

3.1 Ideia Geral

Para treinar as redes neurais utilizando as estratégias evolutivas, a abordagem adotada neste trabalho foi a de usar os pesos das conexões das redes neurais como variáveis de decisão do processo de otimização. Além disso, considerando que o objetivo da otimização é maximizar o desempenho das redes neurais nos jogos selecionados, estes atuaram como função-objetivo da estratégia evolutiva. Dessa forma, os conjuntos de pesos que apresentarem os melhores resultados durante os jogos serão priorizados e reproduzidos pela estratégia evolutiva ao longo do processo de otimização. Este processo é realizado da seguinte maneira:

1. A estratégia evolutiva gera uma população inicial.
2. Os valores dos genes destes indivíduos são inseridos nos pesos das conexões das redes neurais.
3. O jogo é iniciado e executado de forma que cada rede neural controle um dos personagens do jogo.
4. Ao final da partida[1], as pontuações alcançadas pelos personagens são encaminhadas para a EE, para que sejam utilizadas como uma estimativa da qualidade dos indivíduos.
5. Após todas as aptidões terem sido definidas, a estratégia evolutiva segue seu fluxo normal de funcionamento (seleção, clonagem e mutação).
6. Ao final deste fluxo, uma nova geração é criada e o processo retorna para a etapa 2.

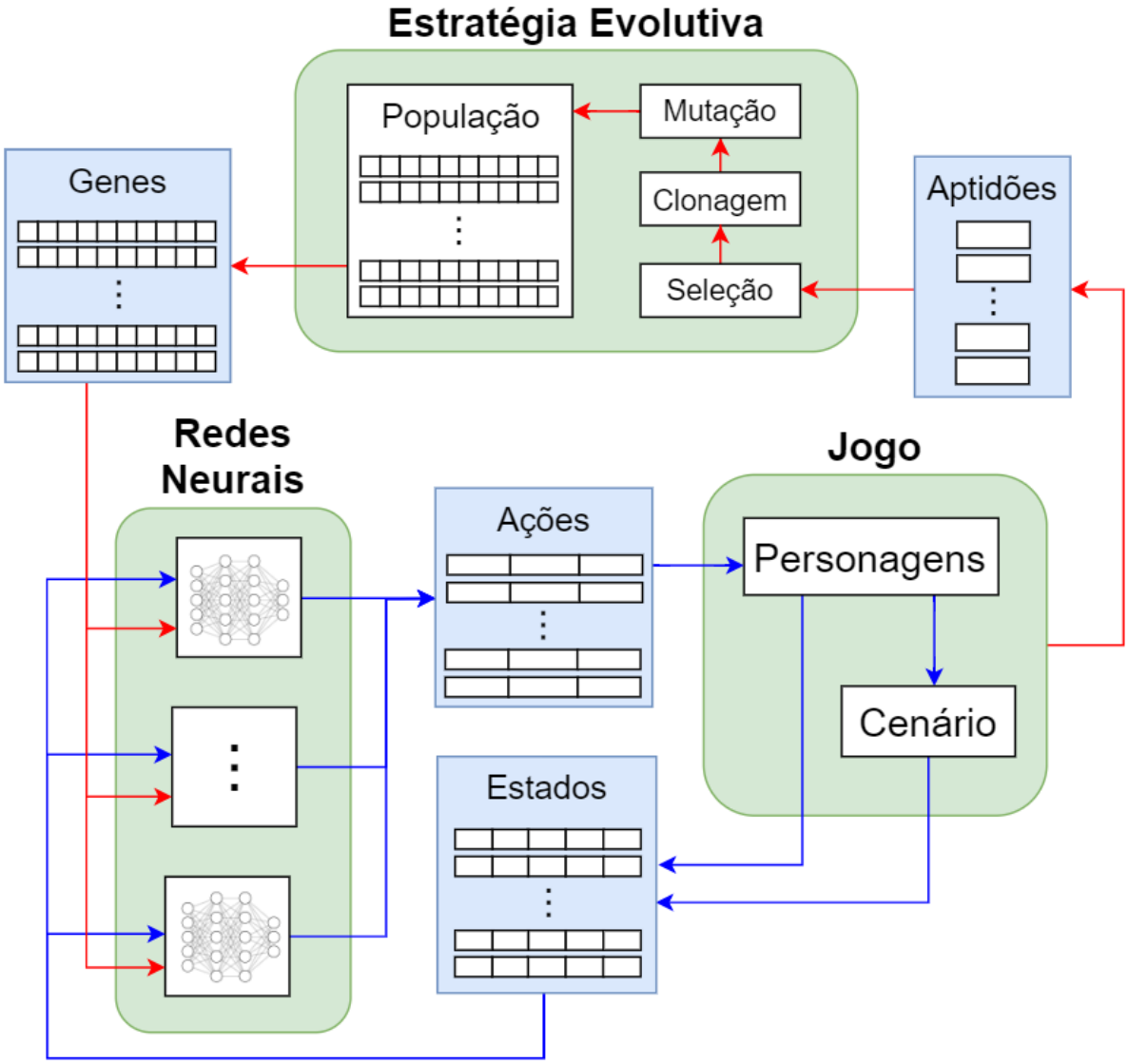
Os detalhes de cada uma destas etapas serão abordados ao longo das próximas subseções. A figura 3.1 exibe um esquema que exemplifica a estrutura e a interação entre as três entidades utilizadas neste trabalho(EE, RNA e Jogo).

Na figura 3.1, as setas vermelhas representam as operações que são realizadas apenas no início e no final da geração. As setas azuis representam as operações que são

[1] O final da partida pode ser definido como o momento em que todos os indivíduos perdem o jogo ou o momento em que um limite de tempo é atingido. O limite de tempo é uma maneira de encerrar a partida dos jogos que permitem que o jogador fique parado e nunca perca.

realizadas constantemente durante toda a execução da partida. Na figura 3.1, é possível perceber que o jogo precisa fornecer algumas informações sobre seu estado para as redes neurais, para que elas possam utilizar essas informações como entrada e assim determinar as saídas que controlarão os personagens do próprio jogo.

Figura 3.1: Interação entre entidades



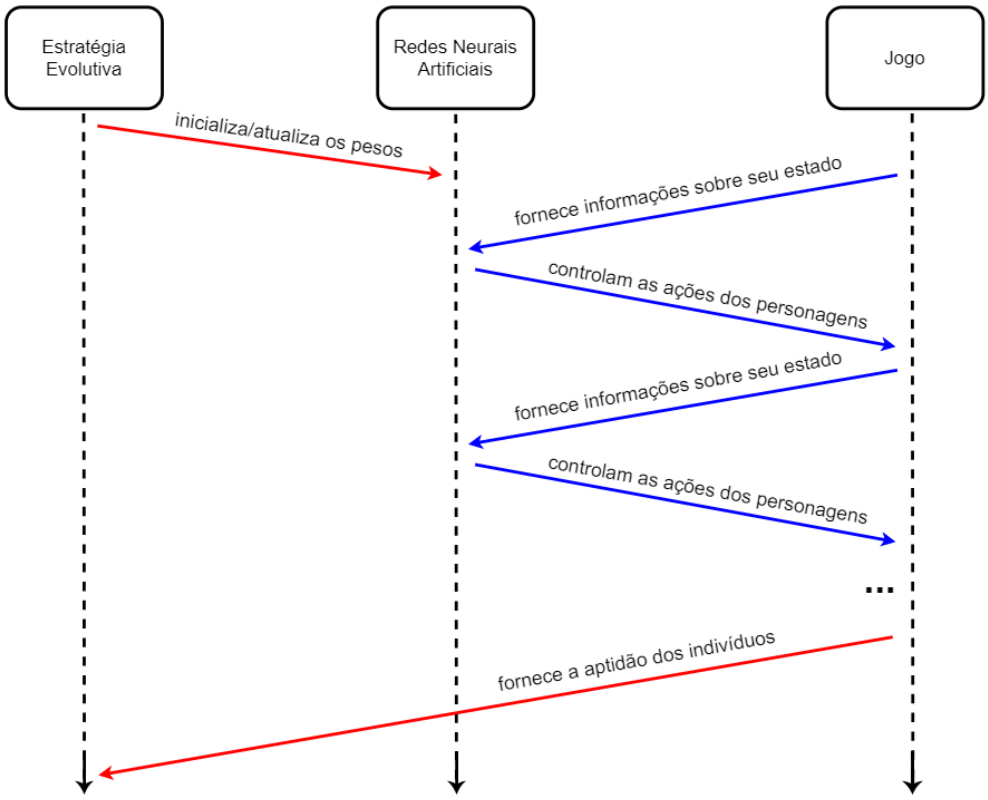
Fonte: O autor

A estratégia evolutiva é a responsável por definir os valores dos pesos das redes neurais(através dos genes). As redes neurais são responsáveis por controlar os personagens do jogo(através das ações). O jogo é responsável por fornecer os valores de entrada para as redes neurais(através dos estados) e a aptidão dos indivíduos para a estratégia evolutiva.

A figura 3.2 exibe um esquema com a sequência das operações explicitada em um eixo temporal. O eixo vertical representa o tempo. Na figura 3.2, pode-se perceber

que as redes neurais e o jogo permanecem trocando informações (setas azuis) até que o personagem perca o jogo e a partida seja encerrada.

Figura 3.2: Interação entre entidades (ao longo do tempo)



Fonte: O autor

Vale ressaltar que, a estratégia evolutiva é um método de otimização estilo caixa preta. Isto significa que ela não possui conhecimento sobre o domínio no qual está atuando e também não utiliza informações específicas sobre ele durante o processo de otimização. Em outras palavras, a estratégia evolutiva não "sabe" que está otimizando os pesos de uma rede neural, e nem que as aptidões dos seus indivíduos são, na verdade, pontuações em um jogo.

Por fim, todas as implementações foram realizadas nas linguagens C/C++ e todos os números aleatórios utilizados durante o trabalho foram gerados com a biblioteca GSL[1] (*GNU Scientific Library*).

[1]Disponível em: `https://www.gnu.org/software/gsl/`

3.2 Estratégias Evolutivas

Neste trabalho, as estratégias evolutivas foram utilizadas de modo a evoluir os pesos de redes neurais. O objetivo desta evolução é produzir redes que sejam capazes de jogar os jogos digitais selecionados.

A seguir serão descritos os detalhes de como a Estratégia Evolutiva foi aplicada ao problema e a forma como foi implementada. Nesta seção também será proposto um método de mutação alternativo ao método originalmente proposto por Rechenberg[20]. Este método foi nomeado de "Mutações Parciais" e no Capítulo 4 será apresentada uma comparação entre ele e o método de Rechenberg.

3.2.1 Genes

Os genes dos indivíduos foram armazenados utilizando valores reais (*double precision*). Os indivíduos que compõem a população inicial tiveram seus genes inicializados de acordo com uma distribuição uniforme no intervalo $[-1, 1]$. Após inicializados, os valores dos genes só foram modificados pelo operador de mutação. Vale relembrar que, estes valores serão utilizados como pesos nas redes neurais e, por isso, a quantidade de genes dos indivíduos depende da quantidade de pesos da rede neural, que por sua vez, depende da topologia da rede.

3.2.2 Mutação

Foram implementados dois métodos de mutação: o método originalmente proposto por Rechenberg[20] e o método denominado "Mutações Parciais" (que será descrito na próxima subseção).

O método de Rechenberg[20] utiliza números aleatórios que seguem uma distribuição normal. Para gerar tais números, foram utilizadas as regras de transformação de Box e Muller[19]. Estas regras demonstram que dois números normalmente distribuídos (com média nula e variância unitária) podem ser obtidos a partir de dois números uniformemente distribuídos (no intervalo [0,1]) da seguinte maneira:

$$x_1 = \sqrt{-2ln(y_1)}sin(2\pi y_2)$$

$$x_2 = \sqrt{-2ln(y_1)}cos(2\pi y_2)$$

onde x_1 e x_2 são os números normalmente distribuídos e y_1 e y_2 são os números uniformemente distribuídos.

Para se obter dois números normalmente distribuídos com média nula mas com variância σ^2, basta multiplicar x_1 e x_2 pelo desvio padrão σ:

$$z_1 = \sigma x_1$$

$$z_2 = \sigma x_2$$

onde z_1 e z_2 são os dois números normalmente distribuídos com média nula e variância σ^2.

3.2.3 Mutações Parciais

O processo de mutação proposto por Rechenberg[20] consiste em adicionar pequenas quantidades aleatórias de ruído em todos os genes do indivíduo. Dessa forma, todos os genes são sutilmente modificados. Por outro lado, a mutação parcial consiste em substituir os valores de alguns poucos genes do indivíduo por valores completamente aleatórios (dentro de um intervalo definido). Dessa forma, apenas alguns genes são modificados, porém, bruscamente.

A motivação por trás da implementação e da avaliação deste método de mutação alternativo é verificar se um processo de mutação que modifica de forma agressiva alguns poucos genes pode apresentar um resultado melhor do que um processo que modifica de forma sutil todos os genes.

As mutações parciais são realizadas da seguinte maneira:

- Primeiramente, é sorteada de forma uniforme e no intervalo $[1, m]$, a quantidade de mutações que o indivíduo sofrerá. O valor m é um parâmetro a ser ajustado e deve ser escolhido com cautela, pois valores muito altos podem dificultar a convergência, enquanto valores muito baixos podem torná-la demasiadamente demorada.

- Após sorteada a quantidade de mutações, os índices dos genes que sofrerão as mutações também são sorteados seguindo uma distribuição uniforme. Para preservar a eficiência computacional do método, qualquer índice pode ser sorteado mais de uma vez.

- Após definidos os índices, os valores dos genes selecionados são substituídos por outros valores aleatórios sorteados de forma uniforme no intervalo $[-a, a]$. O valor a também é um parâmetro a ser definido com cautela, pois ele define o intervalo de valores que os genes poderão assumir e, consequentemente, define o intervalo de valores que os pesos das redes neurais poderão assumir.

  O algoritmo 2 exibe um pseudocódigo para este processo.

Algoritmo 2 Mutações Parciais

1: **Requer:** $vetorGenes$, $quantidadeGenes$, m, a
2: **quantidadeMutacoes** $\leftarrow U(1, m)$
3: **para** $i \leftarrow 0$ **até** $quantidadeMutacoes$ **faça**
4: **indiceGene** $\leftarrow U(0, quantidadeGenes)$
5: **valorGene** $\leftarrow U(-a, a)$
6: $\mathbf{vetorGenes_{indiceGene}} \leftarrow valorGene$
7: **fim para**
8: **retorne** $vetorGenes$

Neste trabalho, o valor a foi fixado em 1 e o valor m foi alvo de testes no Capítulo 4.

3.2.4 Condição de Parada

A condição de parada utilizada foi a condição "tempo máximo de execução". Essa condição estabelece que após esgotada uma determinada quantidade de tempo, o processo de otimização (o treinamento da rede neural) seja interrompido. A quantidade exata de tempo limite variou de acordo com o experimento realizado. Os valores serão apresentados no Capítulo 4.

3.2.5 Seleção

Os dois tipos de seleção apresentados no capítulo 2 (EE-(μ, λ) e EE-$(\mu + \lambda)$) foram implementados e experimentados. Suas performances serão comparadas no Capítulo 4.

3.2.6 Função-Objetivo

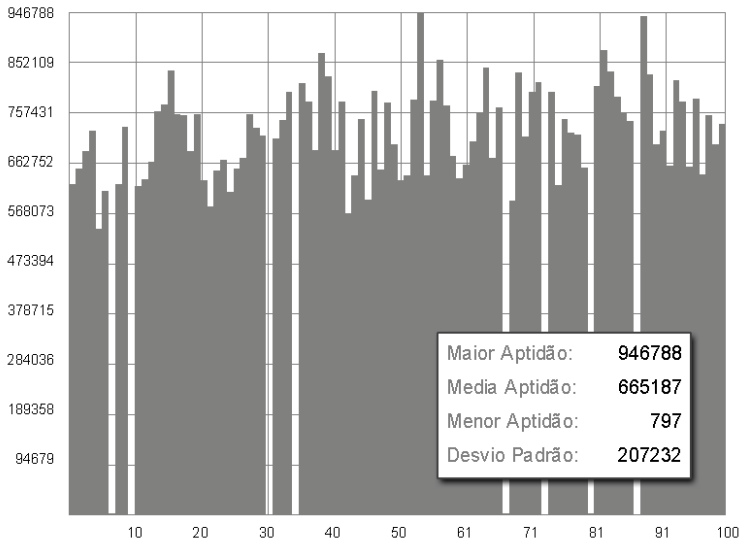
Neste problema, a função-objetivo não possui uma expressão explícita. Por isso, a aptidão dos indivíduos foi calculada utilizando uma simulação computacional (o jogo).

Os personagens (que são controlados pelas redes neurais) foram colocados para jogar e, ao final, da partida atingiram uma determinada pontuação. Esta pontuação foi utilizada para avaliar a habilidade de cada indivíduo no jogo em questão. Porém, uma medida adicional foi aplicada.

Nos jogos desenvolvidos neste trabalho, os cenários são gerados de forma aleatória a cada nova partida. Por isso, jogar apenas uma única partida pode não ser o suficiente para avaliar a habilidade do indivíduo. Em uma única partida, o indivíduo pode não ter sido exposto a uma quantidade significativa de situações para que a sua pontuação represente, de fato, a sua habilidade. Na prática, utilizar como aptidão a pontuação de apenas uma única partida pode fazer com que a generalização da rede fique prejudicada.

A figura 3.3 exibe um gráfico que demonstra a performance de um indivíduo na etapa de validação. Este indivíduo foi obtido em um processo de treinamento onde a população utilizava como aptidão a pontuação de apenas uma única partida (por geração). Na figura 3.3, pode-se perceber que a pontuação atingida em algumas partidas foi muito menor do que a média. Isto demonstra que existem algumas situações específicas nas quais a rede não responde da maneira correta e acaba perdendo o jogo. Ou seja, a rede não está generalizada.

Figura 3.3: Gráfico de validação: pontuação por partida



Fonte: O autor

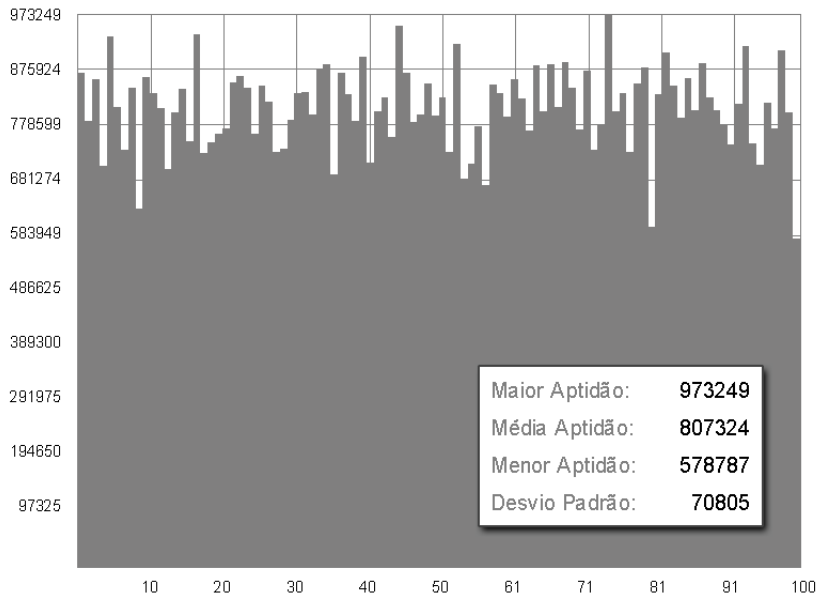
Se o valor retornado pela função-objetivo não representar a habilidade do indivíduo de forma confiável, então todo o processo evolutivo pode estar comprometido. Pois o processo pode acabar não selecionando os indivíduos que realmente aprenderam a jogar.

Visando superar este problema, a Lei dos Grandes Números[34] foi aplicada ao cálculo da aptidão dos indivíduos.

A Lei dos Grandes Números é um teorema fundamental da teoria da probabilidade que foi apresentada, primeiramente, por Jacob Bernoulli, em 1713, no trabalho *Ars Conjectandi.*[35] Esta lei afirma que, se uma experiência aleatória for repetida muitas vezes, a média dos resultados observados se aproximará do valor esperado daquela experiência. Em outras palavras, se uma moeda justa for lançada muitas vezes, a proporção entre cara e coroa se aproximará de 50%, à medida que a quantidade de lançamentos aumenta. Portanto, considerando cada partida jogada pelo indivíduo como uma experiência aleatória, os indivíduos foram colocados para jogar várias partidas antes de terem suas aptidões definidas. O valor utilizado como aptidão foi, então, a média das pontuações obtidas nas n partidas jogadas pelo indivíduo.

A figura 3.4 exibe um gráfico que demonstra a performance de um indivíduo na etapa de validação. Desta vez, porém, o indivíduo foi obtido em um processo de treinamento onde a população utilizava como aptidão a média das pontuações de 25 partidas. Este treinamento foi realizado nas mesmas circunstâncias e com os mesmos parâmetros que o treinamento da figura 3.3 (exceto a quantidade de partidas jogadas pelos indivíduos). Pode-se notar uma queda no valor do desvio padrão e um aumento na robustez, pois agora a rede não apresenta partidas com pontuações muito baixas.

Figura 3.4: Gráfico de validação: Pontuação por Partida



Fonte: O autor

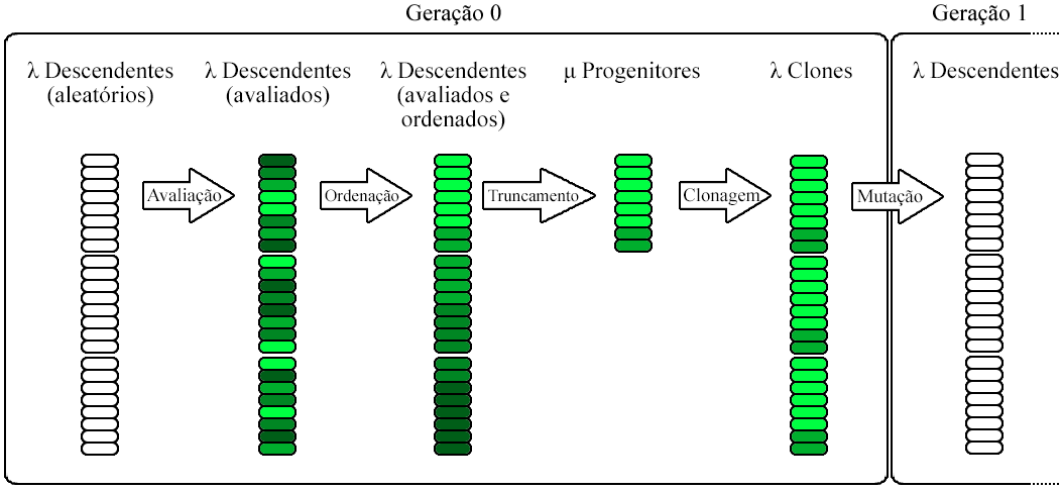
Além da generalização, existe outro benefício em colocar os indivíduos para jogar mais de uma partida por geração: a rede é treinada e testada ao mesmo tempo. Colocar os indivíduos para jogar várias partidas antes de se calcular a aptidão, faz com que as métricas de avaliação obtidas durante o treinamento, sejam muito parecidas com as obtidas durante a validação. Isto acontece pois, essencialmente, o treino e o teste estão executando o mesmo procedimento: expor o indivíduo a várias partidas com situações aleatórias. Contudo, utilizar esta abordagem também faz com que o treinamento demore consideravelmente mais, uma vez que os indivíduos precisarão jogar várias partidas em vez de apenas uma.

A quantidade exata de partidas que os indivíduos jogaram na etapa de treinamento variou de acordo com o jogo. Os valores serão apresentados no Capítulo 4.

3.2.7 Implementação

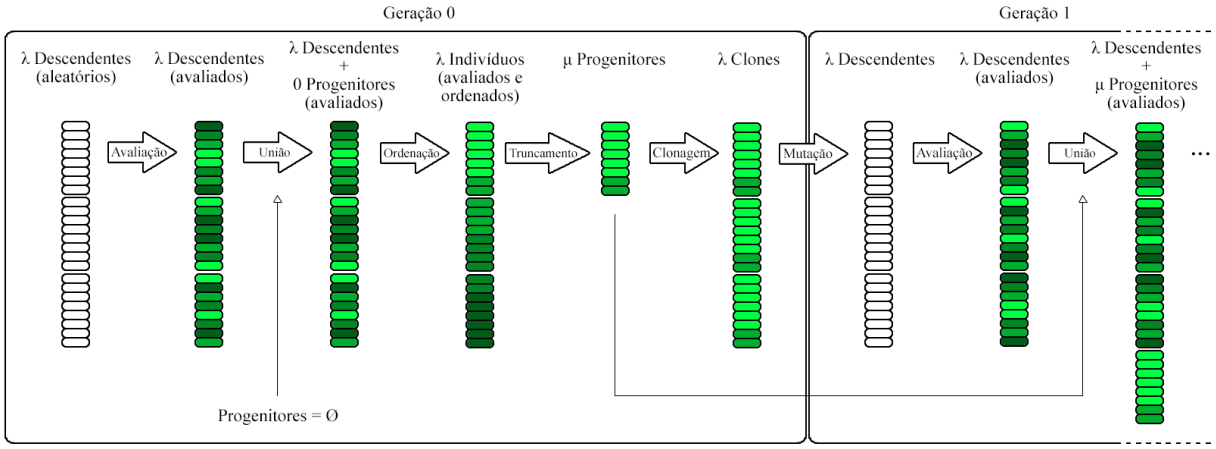
Visando simplificar a integração entre a execução da estratégia evolutiva e o processamento dos jogos, o ponto de partida do processo evolutivo foi modificado. Esta modificação foi realizada de forma a não afetar o funcionamento ou o desempenho do processo. As figuras 3.5 e 3.6 exibem, respectivamente, o funcionamento das estratégias evolutivas EE-(μ,λ) e EE-(μ+λ) após as modificações. Na versão de Rechenberg (exibida nas figuras 2.4 e 2.5), o primeiro passo é a avaliação da aptidão dos progenitores. Na versão deste trabalho, o primeiro passo é a avaliação da aptidão dos descendentes (os indivíduos da população inicial já são considerados "descendentes").

Figura 3.5: Funcionamento modificado EE-(μ,λ)



Fonte: O autor

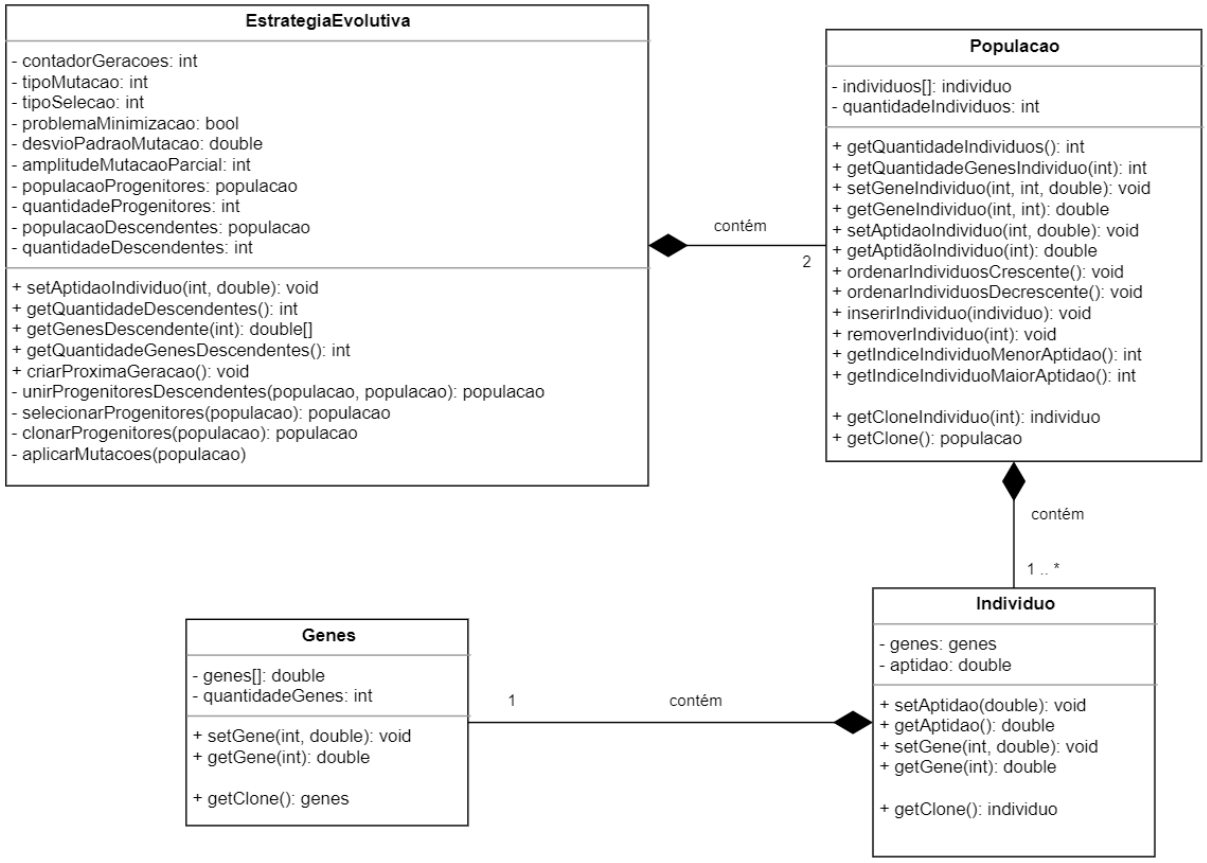
Figura 3.6: Funcionamento modificado EE-(μ+λ)



Fonte: O autor

A figura 3.7 exibe um diagrama de classes da implementação das estratégias evolutivas realizada neste trabalho.

Figura 3.7: Diagrama de classes das estratégias evolutivas

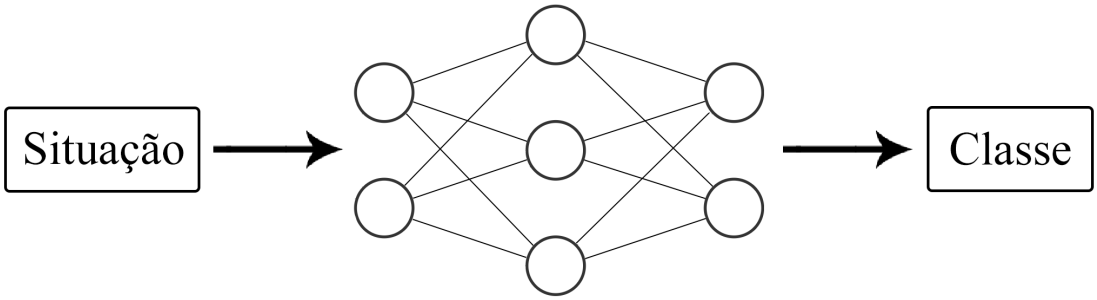


Fonte: O autor

3.3 Redes Neurais Perceptron

As Redes Neurais são modelos amplamente utilizados na literatura por sua eficácia em problemas de classificação.[36][37][38][39] Neste trabalho, as redes neurais também atuaram como um classificador. As redes foram aplicadas de forma a classificar as possíveis situações que o personagem pode experienciar durante o jogo. A figura 3.8 exibe um esquema que demonstra essa relação.

Figura 3.8: Rede neural classificando situações



Fonte: O autor

As possíveis situações que um jogo pode apresentar são formalmente denominadas "estados".

3.3.1 Estados

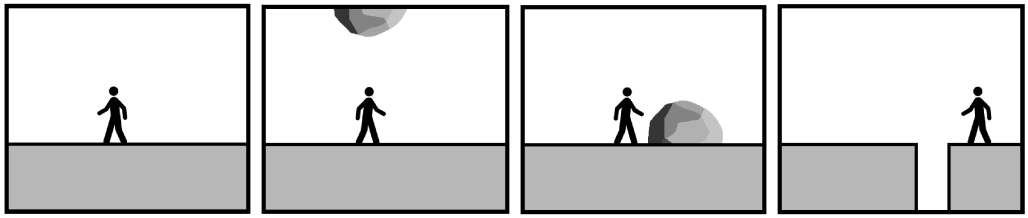
O estado do jogo pode ser definidos da seguinte maneira:

$$estado = (x_1, x_2, x_3, ..., x_n)$$

onde $x_1, x_2, x_3, ..., x_n$ são as informações que caracterizam aquele instante do jogo.

O estado é um conjunto de informações que descrevem as características de cada momento específico do jogo e que, quando agrupadas, diferenciam este momento de qualquer outro. A figura 3.9 exibe alguns estados de um jogo hipotético. Pode-se perceber que, as diferenças entre os estados exibidos são: a posição do personagem, o sentido para o qual ele está olhando, a posição da pedra, a existência (ou não) de um buraco no chão e a posição do buraco (caso ele exista). Estas informações devem estar presentes no vetor de estados, seja diretamente ou indiretamente, uma vez que caracterizam a forma como o jogo está.

Figura 3.9: Estados de um jogo hipotético



Fonte: O autor

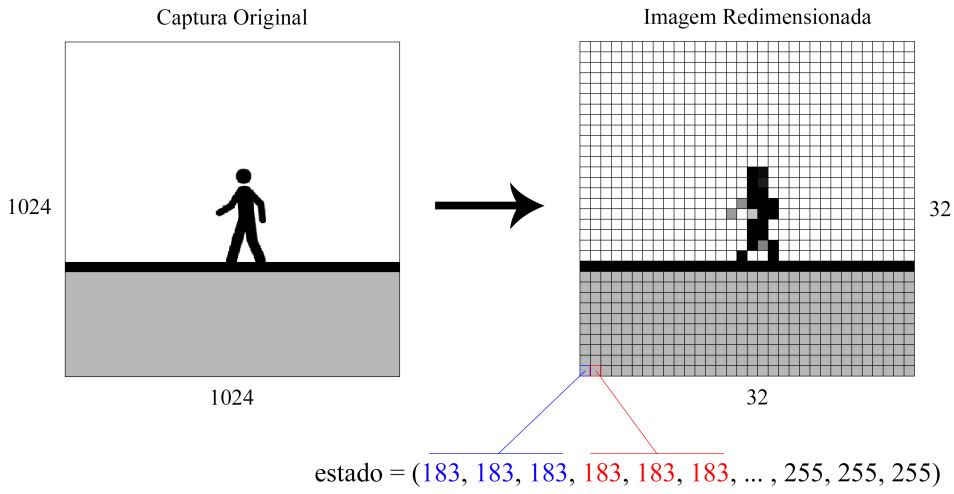
Uma das formas utilizadas na literatura para se representar o estado do jogo é a captura de tela.[12] Nesta abordagem, o vetor que representa o estado é composto pelos valores das cores dos *pixels* da imagem gerada pela captura. Normalmente, essas cores são codificadas no formato RGB (*red, green, blue*). Assim, o vetor de estados pode ser definido por:

$$estado = (R_1, G_1, B_1, R_2, G_2, B_2, ..., R_n, G_n, B_n)$$

onde n é a quantidade de *pixels* da imagem.

Visando melhorar a eficiência do aprendizado, é comum que a imagem resultante da captura seja comprimida antes de ser encaminhada para a rede neural. Isto é feito com o propósito de diminuir a quantidade de *pixels* e, por consequência, a quantidade de valores de entrada da rede neural. A figura 3.10 exibe uma captura de tela com resolução de 1024x1024 *pixels* que foi redimensionada para uma imagem de 32x32 *pixels*. Em seguida, as cores dos seus *pixels* foram utilizadas no vetor que representa o estado do jogo.

Figura 3.10: Captura de tela redimensionada



Fonte: O autor

No entanto, é importante mencionar que a compressão da imagem pode provocar perda de informações. Neste caso, a imagem comprimida deixa de representar o verdadeiro estado do jogo e passa a representar uma aproximação.

Essa forma de modelar o estado do jogo possui alguns aspectos positivos:

- Ao utilizar todos os *pixels* da tela para representar o estado do jogo, todas (ou quase todas) as informações relevantes do jogo estão, mesmo que de forma indireta, contidas na representação.

- Como todos os *pixels* da tela estão sendo usados para definir os estados, nenhuma decisão foi tomada a respeito de quais informações irão compor o vetor de estados. Isto elimina o viés humano na modelagem do estado do jogo.

Estes dois aspectos reduzem a probabilidade de que o vetor que representa o estado do jogo seja modelado de forma insuficiente ou incorreta (isto é, faltar informações importantes para definir unicamente cada momento do jogo). No entanto, essa abordagem também possui alguns aspectos negativos:

- Devido à quantidade elevada de variáveis que compõem o vetor de estados, a quantidade de pesos que a rede neural possuirá também será elevada (visto que os neurônios da primeira camada se conectam com todos os valores de entrada), tornando a rede mais complexa e o treinamento mais difícil e computacionalmente mais custoso.

- Por se tratar de uma imagem, a rede terá que aprender a reconhecer os objetos presentes na imagem (personagens, obstáculos e inimigos) antes de aprender a classificá-los. Esta etapa adicional aumenta a complexidade do processo e pode dificultar o treinamento. Para este tipo de problema, as Redes Neurais Convolucionais[40] são recomendadas. Pois elas possuem técnicas especialmente desenvolvidas para lidar com imagens.

- Por utilizar todos os *pixels* da tela para definir o estado, este pode ficar desnecessariamente complexo, possuindo informações desnecessárias ou redundantes, e que podem, também, dificultar o treinamento da rede.

- Alguns jogos possuem, por exemplo, uma funcionalidade denominada “inventário” (uma mochila onde os personagens guardam itens). Normalmente, o inventário não

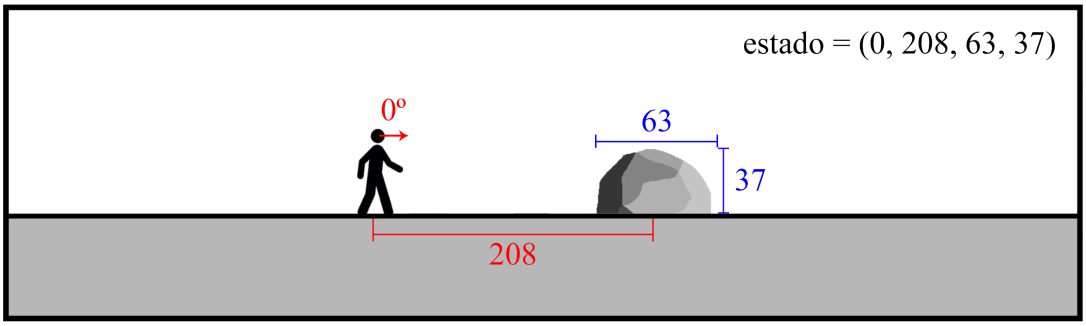
é exibido na tela durante o jogo (é necessário apertar alguns botões para acessá-lo). Por isso, o inventário acaba sendo excluído do vetor de estados do jogo (pois não está presente na captura de tela), sendo necessário incluí-lo de outra maneira.

Devido ao fato de que, neste trabalho, precisou-se executar várias redes neurais simultaneamente (uma para cada indivíduo da estratégia evolutiva) e em tempo real (isto é, várias vezes por segundo), o formato do vetor de estados foi escolhido de forma a priorizar a eficiência computacional[1]. Por isso, a abordagem que utiliza os *pixels* da tela não foi aplicada neste trabalho, pois torna a rede mais complexa (muitos neurônios e muitos pesos). Em vez disso, uma abordagem empírica foi preferida e aplicada. Esta abordagem visa diminuir a quantidade de variáveis no vetor de estados do jogo e, consequentemente, a quantidade de pesos que a rede neural possui, tornando a etapa de treinamento mais rápida.

A abordagem empírica consiste em escolher de forma empírica quais informações serão utilizadas no vetor que representa o estado do jogo. Na prática, o desenvolvedor da rede neural tem de observar e analisar o jogo para definir quais informações são relevantes para representar o estado. Ao contrário do método por *pixels*, esta abordagem deve ser realizada de forma "manual" e individual para cada jogo em questão, pois cada jogo possui uma estrutura e um funcionamento diferente. Normalmente, as informações relevantes para compor o vetor de estados do jogo são aquelas que variam com o tempo (posições dos personagens, velocidades, tamanhos, distâncias etc). A figura 3.11 exibe um exemplo desta abordagem. Na figura 3.11, as dimensões da pedra, a distância entre ela e o personagem principal e a direção para a qual o personagem está olhando foram escolhidas para compor o vetor de estados. Para que a abordagem empírica seja viável, o desenvolvedor da rede neural deve ter acesso ao código do jogo ou a algum tipo de API (*Application Programming Interface*), para que seja possível obter as informações dos elementos do jogo e assim encaminhá-las para a entrada da rede neural.

[1]Para se ter noção, considerando uma população com 1000 indivíduos, cada indivíduo possuindo uma rede neural associada, e o jogo possuindo uma taxa de atualização de 60 quadros por segundo, a quantidade de *feedforwards* realizados será em torno de 1000*60 = 60.000 por segundo. Por isso, qualquer aumento na complexidade da rede produz um grande impacto no desempenho geral do treinamento.

Figura 3.11: Estado de um jogo hipotético modelado de forma empírica



Fonte: O autor

Para fins de comparação, se os estados exibidos na figura 3.9 fossem modelados usando uma captura de tela com uma resolução de apenas 32x32 *pixels*, a quantidade total de *pixels* que a imagem possuiria seria de 1.024. Considerando também que cada pixel possui três valores de cores (RGB), a quantidade total de valores de entrada que a rede neural teria de receber seria de 3.072. Em contra partida, utilizando a abordagem empírica, os estados poderiam ser representados utilizando apenas sete variáveis: as coordenadas x e y do personagem principal, o sentido para o qual ele está olhando, as coordenadas x e y da pedra, a existência (ou não) do buraco no chão e a coordenada x dele (caso exista). Isto produziria uma diminuição de 99,75% na quantidade de valores de entrada da rede neural.

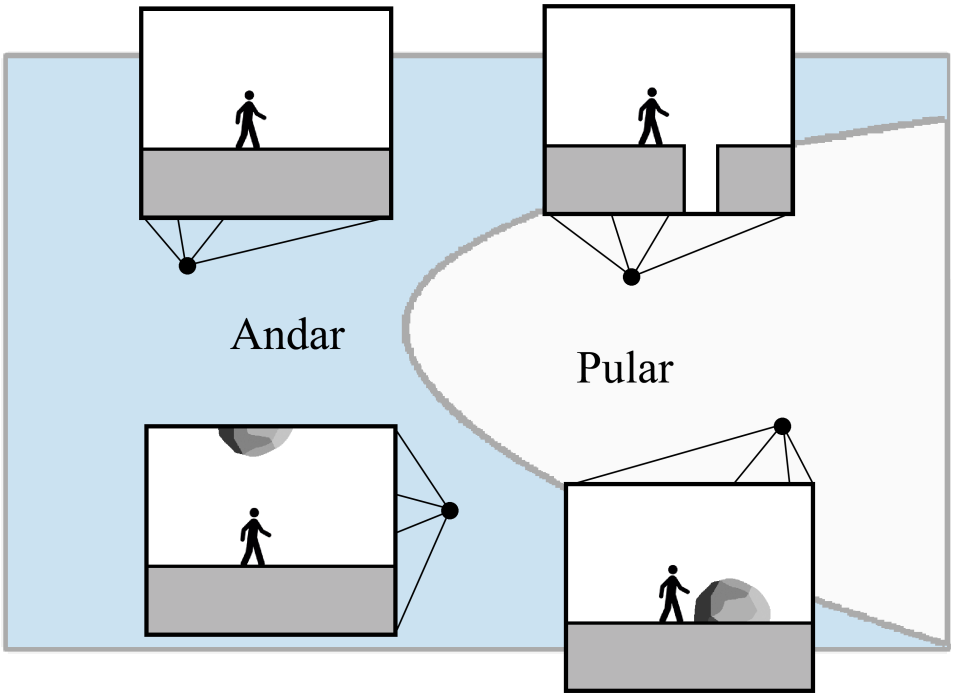
Como mencionado anteriormente, essa forma de modelar o estado do jogo está sujeita ao viés humano, uma vez que um ser humano terá de tomar decisões sobre quais informações (e de que forma) irão compor o estado do jogo. Porém, essa modelagem reduz significativamente a complexidade da rede, tornando o treinamento mais eficiente e justificando, assim, o risco de usá-la.

As informações escolhidas para compor o estado de cada jogo serão descritas nas seções dos respectivos jogos.

3.3.2 Classes

Após definidos os valores de entrada da rede neural, será necessário definir o formato da camada de saída. Na camada de saída, cada neurônio estará associado a uma classe (visto que a rede desempenha o papel de um classificador). Porém, cada classe também estará associada a uma possível ação que o personagem pode executar. Por isso, a saída produzida pela rede neural não apenas classifica a entrada em uma categoria, como também determina uma ação a ser executada. A figura 3.12 exibe o exemplo de uma rede neural classificando os estados de um jogo em duas classes(ações): andar e pular. Cada ponto no plano representa um estado diferente.

Figura 3.12: Plano de estados classificado



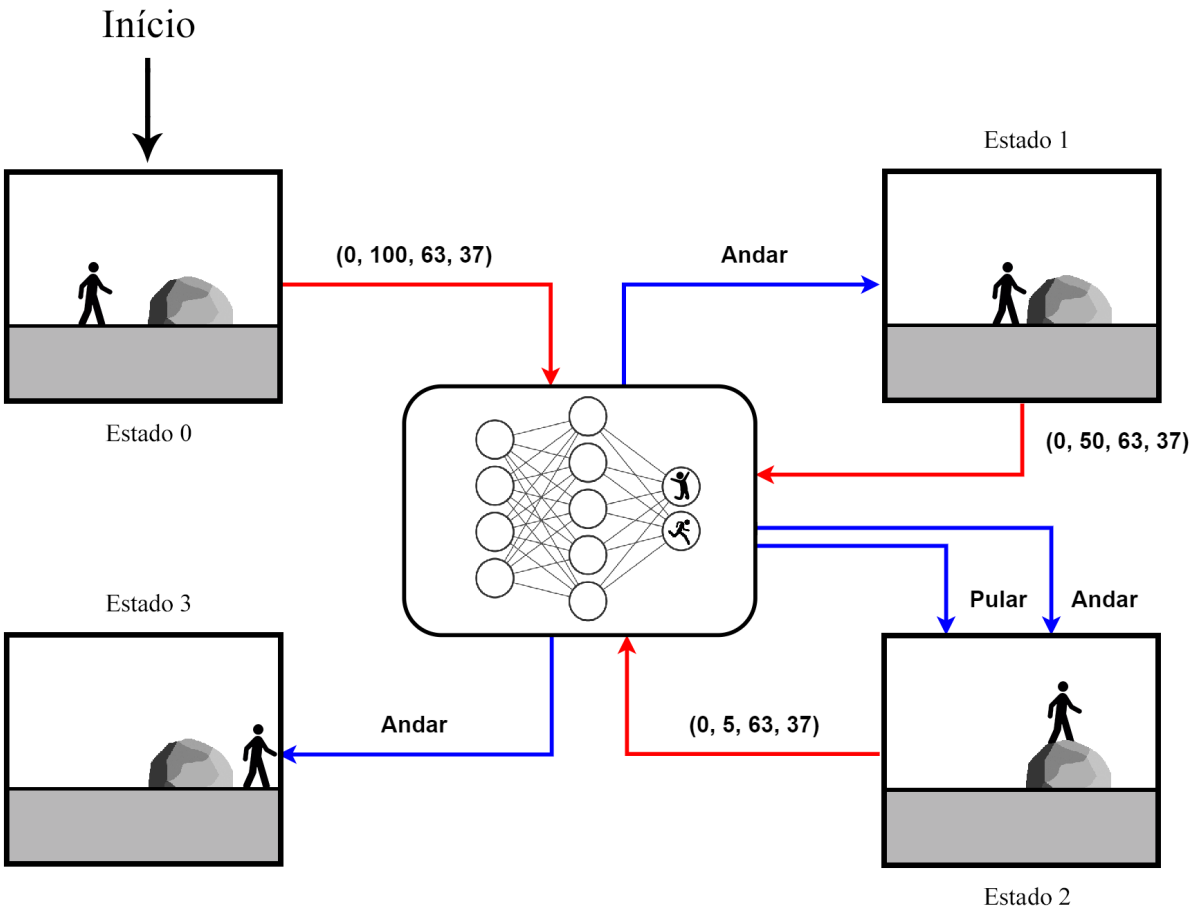
Fonte: O autor

Como as classes estarão associadas com as ações do personagem, a quantidade de classes (e de neurônios na camada de saída) depende da quantidade de ações que o personagem dispõe durante o jogo. Ou seja, haverá um neurônio/classe para cada ação possível.

3.3.3 Execução

Durante a execução do jogo, o estado atual é encaminhado para a rede neural, para que ela determine uma ação. A ação produzida por ela é aplicada ao personagem, fazendo-o sair do estado atual e experienciar um novo estado. Este novo estado, por sua vez, também será encaminhado para a rede, para que uma nova ação seja determinada. Este ciclo se repete até que a partida acabe. É importante notar que, neste caso, o processo de classificação também pode ser interpretado como um processo de tomada de decisão, visto que, a cada situação apresentada, uma ação é realizada. A figura 3.13 exemplifica esse processo.

Figura 3.13: Interação entre o jogo e a rede neural



Fonte: O autor

Existem duas formas de executar as ações com base na saída da rede neural.

1. Todos os neurônios da camada de saída que ativarem terão suas ações associadas executadas: se a função de ativação do neurônio for, por exemplo, a função ReLU ou a função Tangente Hiperbólica, todos os neurônios que apresentarem uma saída maior que zero terão suas ações associadas executadas. Caso a função de ativação seja do tipo sigmoide, serão executadas as ações associadas aos neurônios que apresentarem saídas maiores que 0.5. Vale ressaltar que, nesta abordagem, mais de uma ação pode ser executada pelo personagem na mesma iteração do jogo.

2. Dentre todos os neurônios da camada de saída que ativarem, apenas o neurônio com o maior valor de ativação terá sua ação associada executada: nesta abordagem, apenas uma ação pode ser aplicada ao personagem em uma iteração do jogo, aquela associada ao neurônio que apresentou o maior valor de saída.

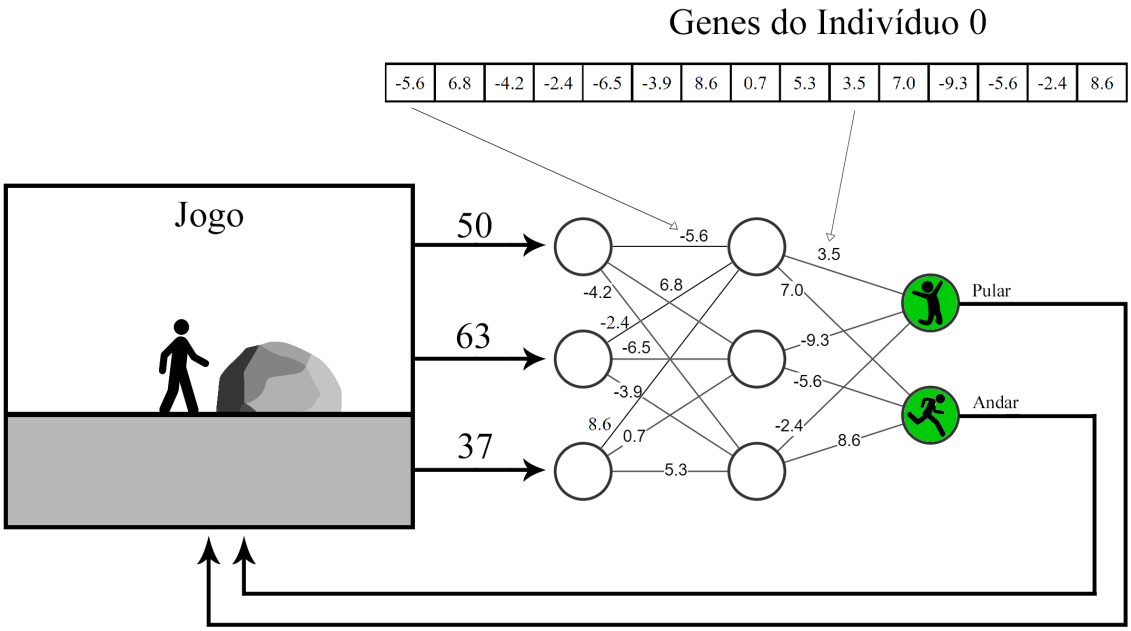
A decisão sobre qual das duas abordagens deve ser utilizada também deve ser feita de forma empírica pelo desenvolvedor da rede neural. Neste trabalho, foi utilizada a abordagem que permite que mais de uma ação seja executada na mesma iteração do jogo.

Vale ressaltar que, em alguns jogos, o personagem pode apresentar ações que são naturalmente opostas (ações que, se executadas ao mesmo tempo, se cancelam), como por exemplo, as ações “acelerar” e “frear” de um carro virtual. Estas ações podem ser consideradas naturalmente opostas, visto que o ato de frear elimina qualquer ganho de velocidade fornecido pelo ato de acelerar. As ações “virar volante para esquerda” e “virar volante para a direita” também podem ser consideradas opostas, pois na prática, é impossível realizar as duas ações ao mesmo tempo. Dessa forma, pode ser benéfico para o desempenho da rede neural que esta restrição esteja imposta no processo, garantindo que, caso estes dois neurônios ativem simultaneamente, apenas uma das duas ações seja executada (aquela associada ao neurônio que apresentou o maior valor de saída dentre os dois envolvidos).

Por fim, a figura 3.14 demonstra com detalhes o modo como as redes neurais deste trabalho se comportam. Nesta figura, os genes do indivíduo 0 são inseridos como pesos na rede neural. Em seguida, o jogo fornece algumas informações sobre o seu estado atual. Após receber essas informações como entrada, a rede realiza o processo de *feedforward* e

determina a saída dos neurônios da camada de saída. As ações associadas aos neurônios que ativaram são executadas no jogo. A cor verde indica que o neurônio ativou.

Figura 3.14: Funcionamento detalhado da rede neural



Fonte: O autor

3.3.4 Implementação

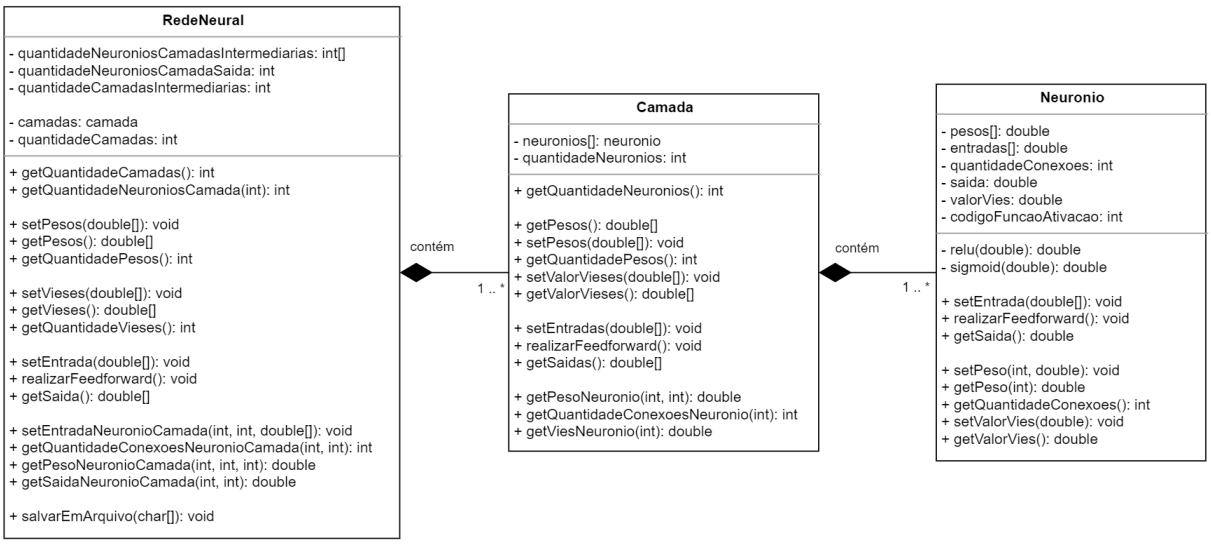
Neste trabalho, todas as redes neurais utilizadas foram do tipo Perceptron, com neurônios completamente conectados e sem retroalimentação. Considerando que o objetivo do trabalho consiste em investigar apenas os parâmetros das Estratégias Evolutivas, as arquiteturas das redes neurais não foram investigadas (isto é, não foram testadas várias arquiteturas distintas), e por isso, alguns aspectos da arquitetura foram fixados em uma determinada configuração para todos os experimentos/jogos. De modo geral, a arquitetura das redes foi estabelecida da seguinte maneira:

- Os valores de entrada das redes neurais serão as informações que compõem o estado do jogo.
- Os valores de entrada das redes neurais foram normalizados para evitar a disparidade entre as escalas.
- As redes possuirão apenas uma camada intermediária.

- A quantidade de neurônios da camada intermediária será exatamente o dobro da quantidade de valores de entrada.
- A camada de saída possuirá um neurônio para cada ação do personagem.
- A função de ativação utilizada nos neurônios da camada intermediária será a função ReLU.
- A função de ativação utilizada nos neurônios da camada de saída irá variar entre ReLU e Sigmoide de acordo com o jogo.

A descrição exata da arquitetura da rede neural utilizada em cada jogo será apresentada nas seções dos respectivos jogos. Por fim, a figura 3.15 exibe um diagrama de classes da implementação das redes neurais.

Figura 3.15: Diagrama de classes das redes neurais



Fonte: O autor

3.4 Jogos

Quatro jogos foram selecionados e implementados "do zero" com o objetivo de serem utilizados como ambiente de avaliação para as Redes Neurais e para as Estratégias Evolutivas. Dentre eles, dois são réplicas de jogos que já existem e dois foram desenvolvidos especificamente para este trabalho. Os jogos tiveram de ser implementados por dois motivos:

1. Para que o autor tenha acesso ao código do jogo e assim viabilize a aplicação da abordagem empírica na modelagem do vetor de estados.

2. Para que seja possível modificar e dificultar o jogo, caso necessário.

Os quatro jogos possuem cenários que são gerados proceduralmente.[41] A geração procedural consiste em utilizar algoritmos(ou algumas regras simples) para criar de forma automática, e em tempo de execução, os cenários dos jogos (terreno, obstáculos, inimigos etc). Normalmente, o processo de geração utiliza números aleatórios para que os cenários gerados tenham variedade e diversidade. No entanto, alguns parâmetros predefinidos também são utilizados para que os cenários façam sentido. Com a geração procedural, cada partida do jogo se torna única, pois seu cenário foi gerado em tempo de execução e utilizando aleatoriedade. Por isto, esse tipo de cenário pode ajudar a reduzir o efeito do Sobreajuste (*overfitting*) nas redes neurais, pois a cada nova partida jogada, novas situações são apresentadas aos indivíduos.

Os jogos foram desenvolvidos de forma a suportar a existência de vários personagens simultâneos no cenário. Dessa forma, todos os indivíduos que compõem a população da Estratégia Evolutiva serão treinados jogando a mesma partida ao mesmo tempo.

Para realizar a implementação da parte gráfica dos jogos, foi utilizada uma biblioteca gráfica denominada "PIG[1]" (Programming Interface for Games). Esta biblioteca possui funções que permitem criar janelas, desenhar nestas janelas e detectar o pressionamento de teclas e botões dos dispositivos de entrada (mouse e teclado).

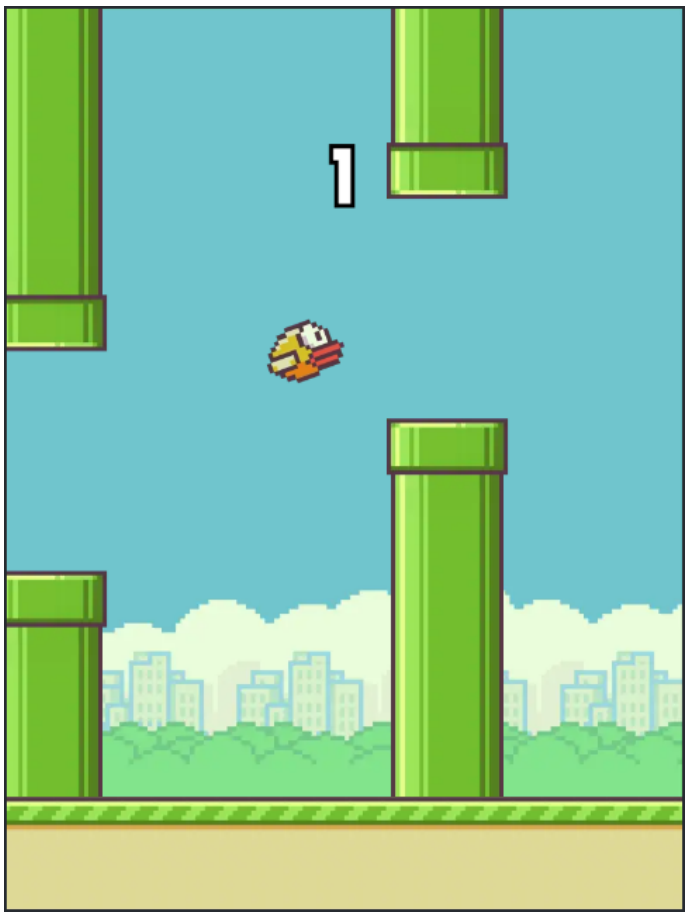
Vale mencionar que os jogos utilizam o *pixel* como unidade de medida de espaço e a iteração como unidade de medida de tempo.

[1]Disponível em: `https://github.com/PIGDevUff/PigDev`

3.4.1 Flappy Bird

No início de 2014, um jogo para *smartphones* denominado "*Flappy Bird*" fez bastante sucesso. Neste jogo, o jogador controla um pássaro que deve atravessar uma sequência de pares de canos, sem encostar neles. Cada par de canos possui uma abertura por onde o pássaro deve passar. A altura dessas aberturas varia ao longo do jogo, tornando a travessia dinâmica e desafiadora. Porém, a real dificuldade do jogo está no modo como o pássaro se movimenta, descrevendo uma parábola negativa. A única ação que o jogador pode executar é o ato de pular, que possui uma altura fixa. O objetivo do jogo é atravessar a maior quantidade de pares de canos possível. A figura 3.16 exibe uma captura de tela do jogo original.

Figura 3.16: Captura de tela do Flappy Bird



Fonte: Flappy Bird[1]

[1]Disponível em: `https://flappybird.io/`

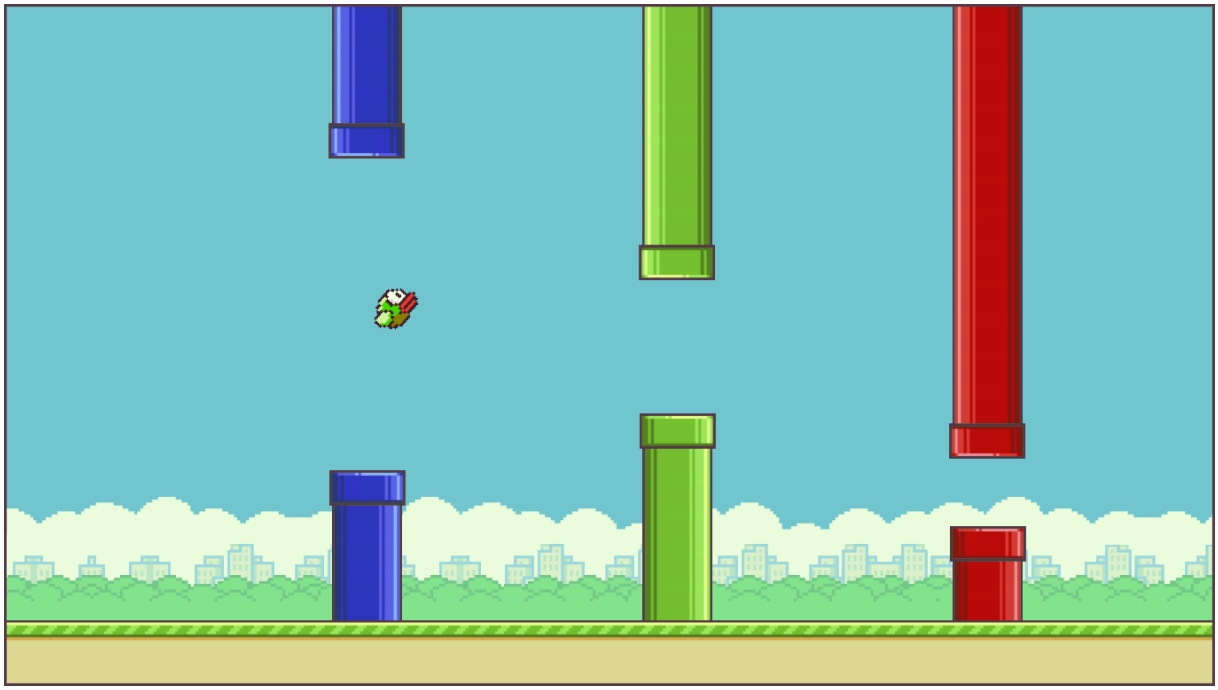
Implementação e Modificações

A versão original deste jogo demonstrou-se consideravelmente fácil para as redes neurais, pois, de acordo com os testes realizados pelo autor, os indivíduos da população inicial já conseguem jogar o jogo de forma a nunca perder. Por isso, neste trabalho, o jogo foi modificado de forma a torná-lo mais complexo, e assim, possibilitar que as diferenças entre os métodos avaliados no Capítulo 4 sejam expostas. As seguintes modificações foram aplicadas ao jogo:

- Um par de canos vermelhos foi adicionado. Estes canos possuem uma abertura muito pequena, impossibilitando a passagem através da ação "pular". Por isto, uma segunda ação foi adicionada ao personagem, a ação de "planar". Ao executar essa ação, o personagem pode voar de maneira estável, descrevendo uma linha reta e caindo mais lentamente em direção ao solo. Para evitar que o jogador abuse desta habilidade, foi inserido um tempo de espera de 100 quadros entre os usos da ação.
- O par de canos verdes foi colocado para se movimentar no eixo vertical. A velocidade de cada um destes pares de canos é aleatória e definida no momento de sua geração.
- Um par de canos azuis foi adicionado. Estes canos possuem uma abertura muito grande, facilitando a passagem. Porém, assim como os canos verdes, eles também se movem no eixo vertical, no entanto, de maneira muito mais veloz.
- Todos os três tipos de canos tem a mesma probabilidade de serem gerados no cenário.
- Ao atingir a quantidade de 100.000 *pixels* percorridos, a partida é encerrada.

A motivação em adicionar estes obstáculos foi a de proporcionar variação e diversidade para as situações do jogo, dificultando a classificação realizada pela rede neural. A figura 3.17 exibe uma captura de tela da versão do jogo implementada e modificada pelo autor.

Figura 3.17: Captura de tela do Flappy Bird (modificado)



Fonte: O autor

A tabela 3.1 apresenta os valores de alguns parâmetros utilizados na implementação do jogo.

Tabela 3.1: Parâmetros Flappy Bird

velocidadeCenario: -2	amplitudeCanoAzul: 350
velocidadePulo: 5	amplitudeCanoVermelho: 75
intensidadeGravidade: -0.3	amplitudeCanoVerde: 150
intervaloVelocidadeCanoAzul: $[-10, 10]$	distanciaEntreCanos: 350
intervaloVelocidadeCanoVerde: $[-2, 2]$	

Modelagem do Estado

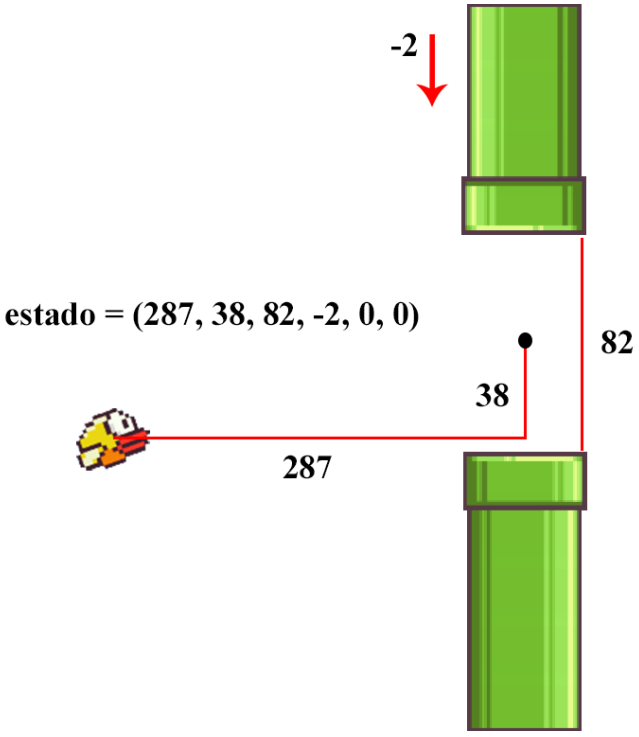
Para este jogo, o vetor de estados foi modelado utilizando as seguintes informações:

- A distância horizontal entre o pássaro e o próximo par de canos. A "distância horizontal" é definida pela diferença entre a coordenada x do par de canos e a coordenada x do pássaro. O "próximo par de canos" é definido como o primeiro par de canos à direita do pássaro.

- A distância vertical entre o pássaro e o centro da abertura do próximo par de canos. A "distância vertical" é definida pela diferença entre a coordenada y do centro da abertura do par de canos e a coordenada y do pássaro.
- O comprimento da abertura entre o próximo par de canos.
- A velocidade vertical do próximo par de canos.
- Um número inteiro que indica se o próximo par de canos é do tipo vermelho. O valor 1 representa o valor verdadeiro e o valor 0 representa o valor falso.
- Um número inteiro que indica se o pássaro está planando. O valor 1 representa o valor verdadeiro e o valor 0 representa o valor falso

A figura 3.18 exibe um exemplo desta modelagem.

Figura 3.18: Modelo do estado do Flappy Bird



Fonte: O autor

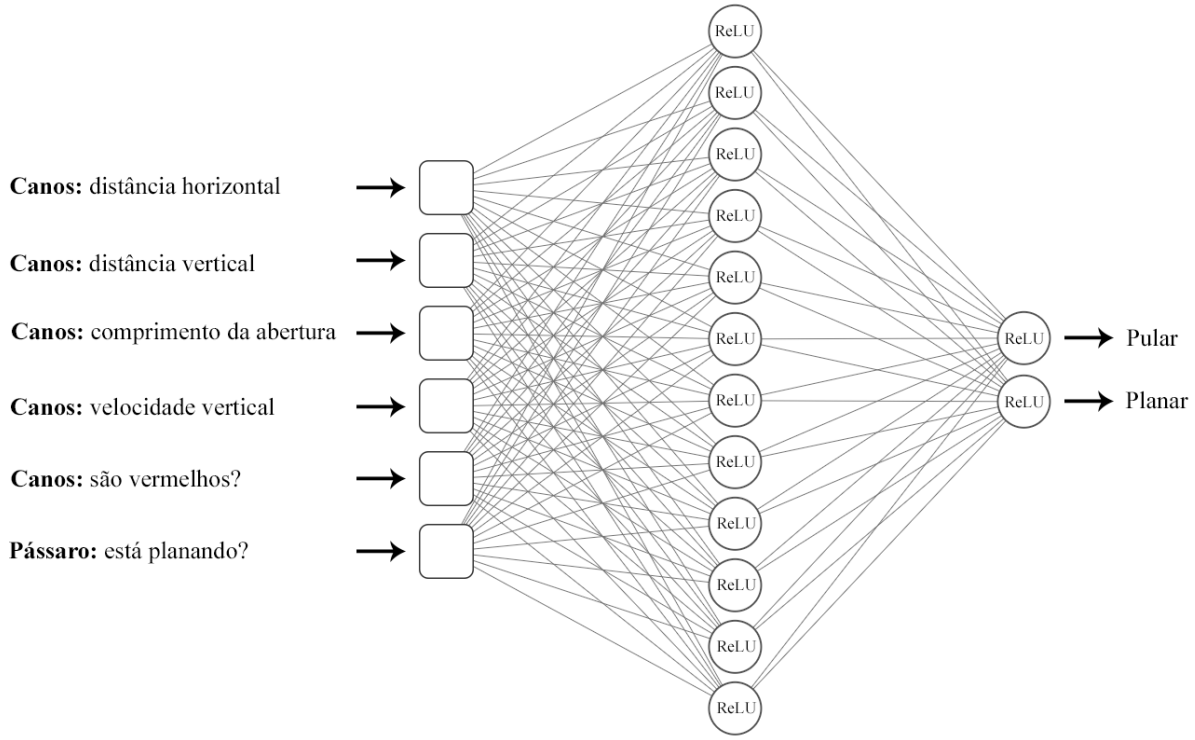
Função-Aptidão

A métrica utilizada como função-aptidão (que deve ser maximizada) para este jogo foi a quantidade de *pixels* percorridos pelo indivíduo. Esta métrica pode ser considerada equivalente à "quantidade de tempo sobrevivido". Essa métrica foi preferida à "quantidade de canos ultrapassados" pois possui maior precisão (é discreta em passos menores).

Arquitetura da Rede Neural

Considerando as seis informações que compõem o estado do jogo e as duas ações que o personagem pode executar, a arquitetura da rede neural utilizada neste jogo ficou definida como mostra a figura 3.19.

Figura 3.19: Arquitetura da RNA utilizada no Flappy Bird



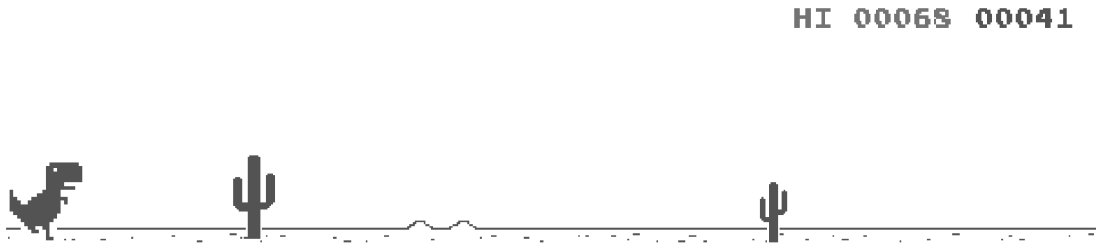
Fonte: O autor

3.4.2 Dinossauro do Google Chrome

Ao apresentar uma falha na conexão com a internet, o navegador de internet *Google Chrome* exibe na tela um jogo. Nele, o jogador controla um dinossauro que deve desviar de uma sequência de obstáculos (cactos e pássaros) que surgem em seu caminho. Cada obstáculo possui suas próprias dimensões (e no caso dos pássaros, uma altura até o solo) tornando o jogo dinâmico. Porém, o verdadeiro desafio está no fato da velocidade do cenário aumentar ao longo do tempo, diminuindo o tempo de reação disponível para o jogador.

O jogador dispõe de duas ações para controlar o movimento do dinossauro: pular e abaixar. A habilidade de abaixar é útil quando o obstáculo é um pássaro (é possível passar abaixado por baixo do pássaro), mas principalmente quando o personagem se encontra no ar, pois, executar esta ação durante o pulo fornece um aumento na velocidade de queda. O objetivo do jogo é desviar da maior quantidade de obstáculos possível. A figura 3.20 exibe uma captura de tela do jogo original.

Figura 3.20: Captura de tela do Dinossauro do Google Chrome



Fonte: Dinossauro do Google Chrome[1]

Implementação e Modificações

A versão original deste jogo demonstrou-se relativamente simples para as redes neurais, pois, apesar da velocidade do cenário aumentar com o tempo, este aumento possui um limite. E uma vez aprendido a jogar no limite, as redes neurais não mais perdem o jogo. Por isso, neste trabalho, o jogo foi modificado e o limite da velocidade do cenário foi retirado. Dessa forma, a velocidade do cenário poderá aumentar "para sempre".

[1]Disponível em: `www.chromedino.com` ou em `chrome://dino` (no navegador Google Chrome)

Esta remoção foi realizada com o objetivo de tornar o jogo mais difícil de ser aprendido, para que as diferenças entre os métodos experimentados no Capítulo 4 sejam expostas. A figura 3.21 exibe uma captura de tela da versão do jogo implementada e modificada pelo autor.

Figura 3.21: Captura de tela do Dinossauro do Google Chrome (modificado)

Fonte: O autor

Na versão do jogo implementada pelo autor, todos os obstáculos possuem a mesma probabilidade de serem gerados no cenário. A figura 3.22 exibe todos os obstáculos possíveis.

Figura 3.22: Obstáculos do jogo do Dinossauro do Google Chrome

Fonte: O autor

A tabela 3.2 apresenta os valores de alguns parâmetros utilizados na implementação do jogo.

Tabela 3.2: Parâmetros Dinossauro do Google Chrome

velocidadeInicialCenario: -5 aceleracaoCenario: -0.001 intensidadeGravidade: -0.4 velocidadePulo: 9	velocidadeAdicionalQueda: -5 distanciaMinimaEntreObstaculos: 600 distanciaMaximaEntreObstaculos: 700

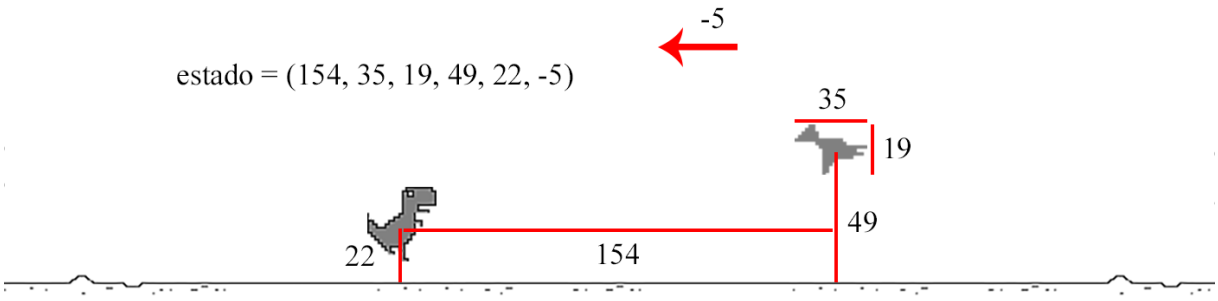
Modelagem do Estado

Para este jogo, o vetor de estados foi modelado utilizando as seguintes informações:

- A distância horizontal entre o dinossauro e o próximo obstáculo. A "distância horizontal" é definida pela diferença entre a coordenada x do obstáculo e a coordenada x do dinossauro. O "próximo obstáculo" é definido como o primeiro obstáculo a direita do dinossauro.
- A largura do próximo obstáculo.
- A altura do próximo obstáculo.
- A coordenada y do próximo obstáculo.
- A coordenada y do dinossauro.
- A velocidade do cenário.

A figura 3.23 exibe um exemplo desta modelagem.

Figura 3.23: Modelo do estado do Dinossauro do Google Chrome



Fonte: O autor

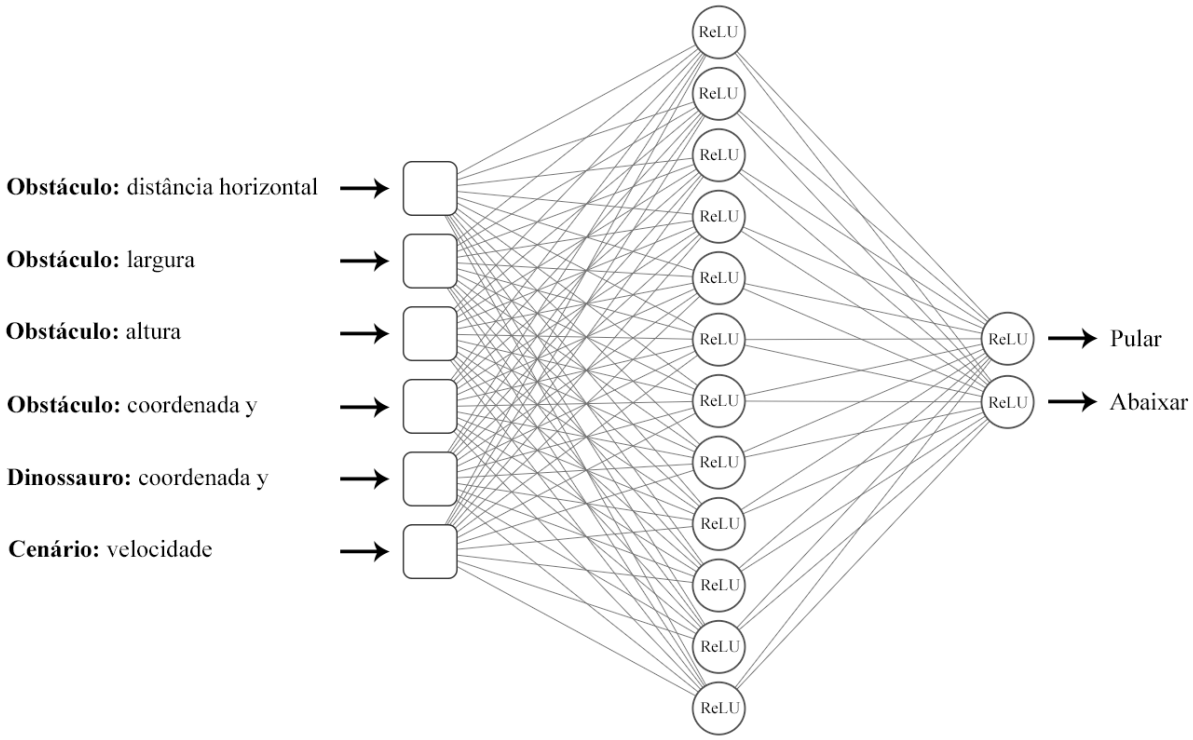
Função-Aptidão

A métrica utilizada como função-aptidão (que deve ser maximizada) para este jogo foi a quantidade de *pixels* percorridos pelo indivíduo. Esta métrica pode ser considerada equivalente à "quantidade de tempo sobrevivido". Essa métrica foi preferida à "quantidade de obstáculos ultrapassados" pois possui uma precisão maior (é discreta em passos menores).

Arquitetura da Rede Neural

Considerando as seis informações que compõem o estado do jogo e as duas ações que o personagem pode executar, a arquitetura da rede neural utilizada neste jogo ficou definida como mostra a figura 3.24.

Figura 3.24: Arquitetura da RNA utilizada no Dinossauro do Google Chrome

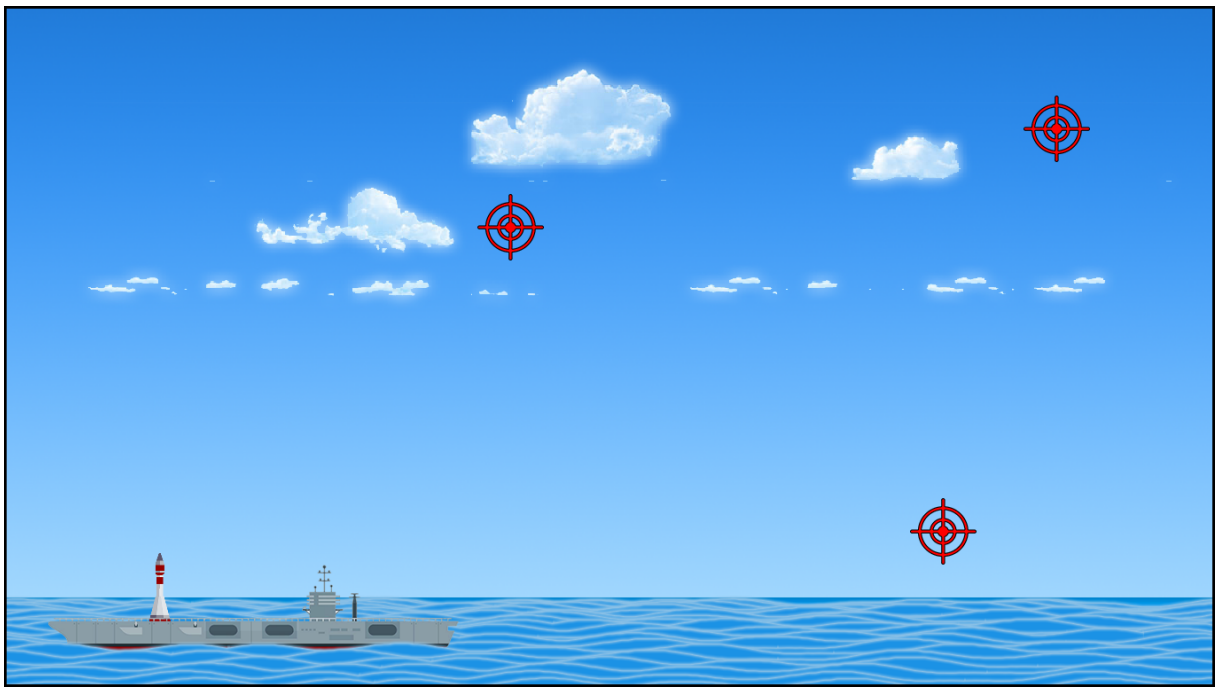


Fonte: O autor

3.4.3 Simulador de Foguete

Neste jogo, o jogador controla um foguete que deve decolar, atingir três pontos específicos do cenário e pousar. O foguete inicia o jogo em estado de repouso em cima de um navio. Após decolar, o jogador deve navegar com o foguete até os três pontos determinados(que têm suas posições definidas aleatoriamente no início do jogo). Após atingir os pontos, o jogador deve retornar o foguete para o navio e pousar. Para realizar o pouso com sucesso, no momento em que o foguete encostar no navio, ele deve estar com uma inclinação em relação ao navio próxima a 90 graus (isto é, entre 88 e 92 graus) e uma velocidade (tanto horizontal, quanto vertical) menor que 5 *pixels* por iteração. A figura 3.25 exibe uma captura de tela do jogo.

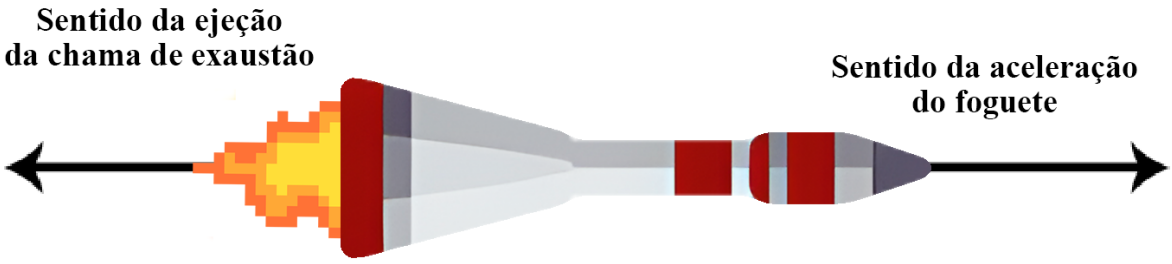
Figura 3.25: Captura de tela do Simulador de Foguete



Fonte: O autor

O real desafio do jogo está em controlar os movimentos do foguete. Na parte inferior do foguete existe um bocal por onde é expelida a chama de exaustão. A chama de exaustão proporciona ao foguete um empuxo que atua no sentido oposto ao movimento dela. A figura 3.26 exibe essa relação.

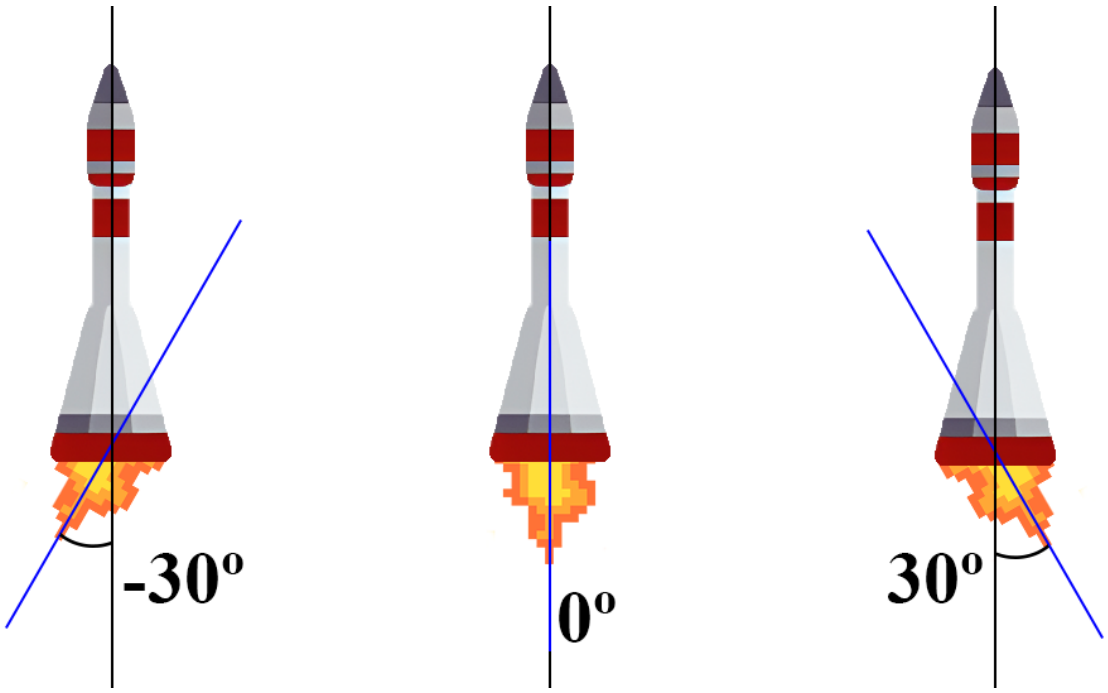
Figura 3.26: Dinâmica de Forças do Foguete



Fonte: O autor

Contudo, para que o foguete consiga controlar a direção do seu movimento, o bocal por onde é expelida a chama de exaustão foi implementado de modo maleável, podendo direcionar a chama para ângulos que variam de -30 até 30 graus em relação ao foguete, como mostra a figura 3.27.

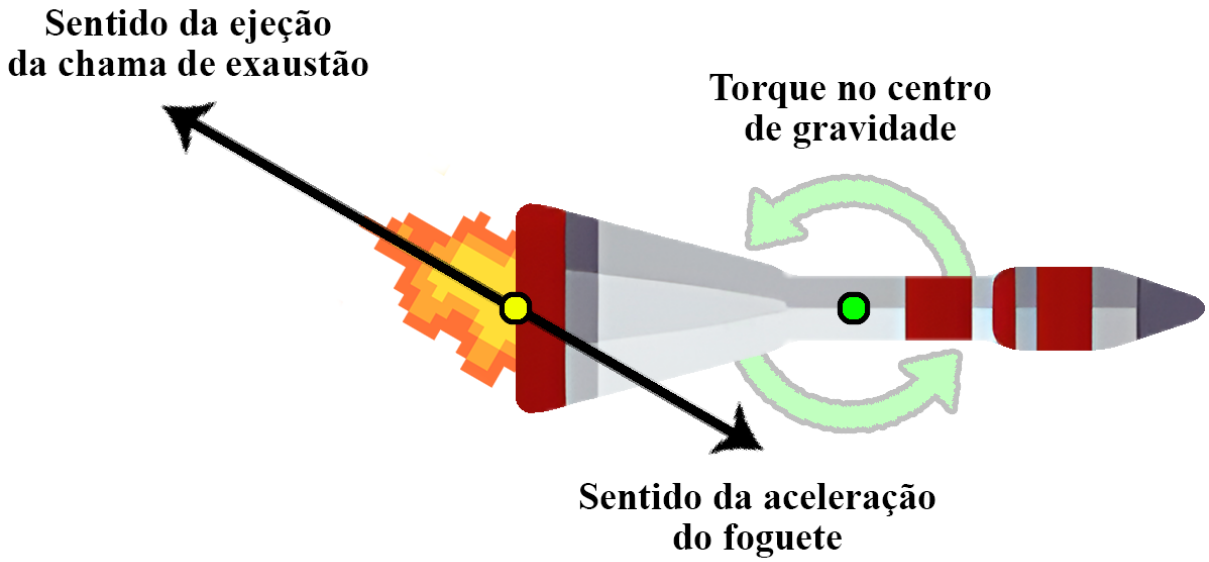
Figura 3.27: Direcionamento da chama de exaustão



Fonte: O autor

Além de controlar a direção da aceleração do foguete, este direcionamento da chama de exaustão também provoca um pequeno torque no foguete, fazendo-o girar em torno do seu centro de gravidade. A figura 3.28 exibe este comportamento.

Figura 3.28: Torque causado pela chama de exaustão



Fonte: O autor

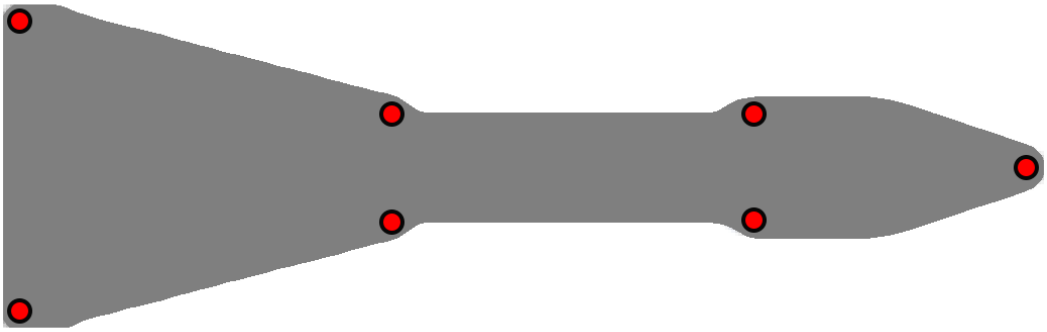
Em um foguete real, a maior parte da massa é composta por combustível. À medida que este combustível vai sendo consumido, a massa diminui e a posição do centro de gravidade do foguete se desloca. Neste trabalho, tanto a existência quanto o consumo de combustível não foram implementados. Ou seja, o foguete possui uma massa constante e uma quantidade de combustível "infinita". O centro de gravidade do foguete foi posicionado no centro geométrico da figura do foguete e permaneceu fixo durante toda a partida. A figura 3.28 exibe este ponto (na cor verde).

Visando manter a simulação simples, porém o mais realista possível, o jogador dispõe de apenas três ações para controlar o movimento do foguete: acelerar, virar o bocal para esquerda e virar o bocal para direita. Cada uma dessas ações possui uma intensidade, que pode variar entre 0 e 1. No caso da aceleração, o valor 0 representa nenhuma aceleração, enquanto o valor 1 representa aceleração máxima(0.3 *pixels* por iteração ao quadrado). No caso das ações que controlam o bocal, o valor 0 representa nenhuma inclinação (0) e o valor 1 representa inclinação máxima (30). Caso as ações "virar bocal para esquerda" e "virar bocal para a direita" sejam executadas ao mesmo tempo, a inclinação efetiva será a soma das inclinações de cada ação. Vale mencionar que, a ação de acelerar é a responsável por criar a chama de exaustão, enquanto as ações de virar o bocal são responsáveis por direcioná-la. Portanto, para que a inclinação do

foguete seja, de fato, alterada, as duas ações (acelerar e virar bocal) devem ser executadas ao mesmo tempo.

Para detectar a colisão do foguete com o navio ou com o mar, foram utilizados sete pontos específicos ao redor do foguete. Quando um destes pontos encosta no navio ou no mar, o foguete é considerado como colidido. A figura 3.29 exibe esses pontos.

Figura 3.29: Pontos de colisão do foguete



Fonte: O autor

A tabela 3.3 apresenta os valores de alguns parâmetros utilizados na implementação do jogo.

Tabela 3.3: Parâmetros do Simulador de Foguete

velocidadeHorizontalMaximaFoguete: 50	aceleracaoGravidade: -0.15
velocidadeVerticalMaximaFoguete: 50	aceleracaoMaximaFoguete: 0.3
larguraCenario: 2732	inclinaçãoMaximaChamaExaustao: 30
alturaCenario: 1536	anguloFogueteMinimoPouso: 88
larguraFoguete: 50	anguloFogueteMaximoPouso: 92
alturaFoguete: 160	velocidadeFogueteMaximaPouso: 5
incrementoAnguloTorque: 1	tempoLimitePartidaDecolar: 1500
larguraNavio: 926	tempoLimitePartidaVoar: 1500
alturaNavio: 216	tempoLimitePartidaPousar: 2000

Modelagem do Estado

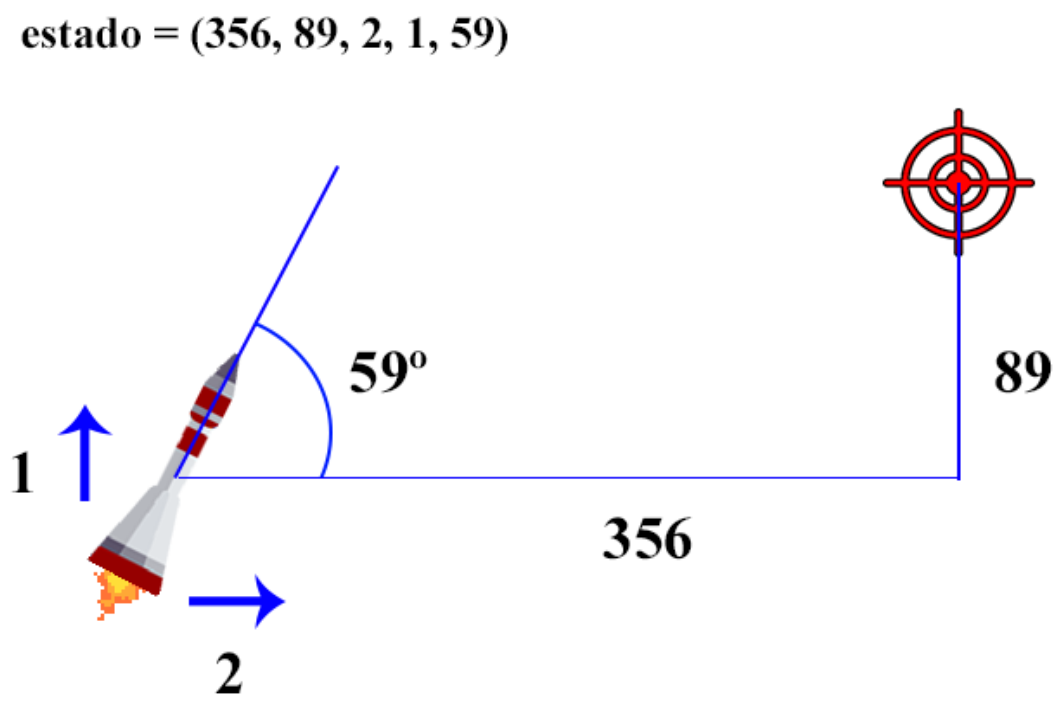
Neste jogo, o vetor de estados foi modelado utilizando as seguintes informações:

- A distância horizontal entre o foguete e o ponto de destino.
- A distância vertical entre o foguete e o ponto de destino.

- A velocidade horizontal do foguete.
- A velocidade vertical do foguete.
- O ângulo do foguete em relação ao eixo horizontal.

A figura 3.30 exibe um exemplo desta modelagem.

Figura 3.30: Modelo do estado do Simulador de Foguete



Fonte: O autor

Treinamento

Para simplificar o treinamento, o jogo foi dividido em três etapas distintas: decolar, voar e pousar. Isto significa que cada indivíduo da estratégia evolutiva jogará três partidas, cada uma com apenas um único objetivo específico(ou decolar, ou voar, ou pousar) e um tempo limite para a execução. A aptidão final será a média das aptidões obtidas nestas três partidas. A figura 3.31 exibe as três etapas.

Figura 3.31: Etapas do Treinamento

(a) Decolar (b) Voar (c) Pousar

Fonte: O autor

A posição do foguete, do navio, do ponto destino e o ângulo inicial do foguete são definidos de forma aleatória no início de cada partida. O ângulo inicial do foguete pode variar de 0 à 360 graus. Portanto, é possível que em alguma partida o foguete comece o jogo “de cabeça pra baixo”.

Função-Aptidão

Neste jogo, a função-aptidão que deve ser minimizada foi definida da seguinte maneira:

$$f(i,p) = distancia_{ip} + 10 * angulo_i + 10 * velocidade_i$$

onde:

- $distancia_{ip}$ representa a distância euclidiana entre o centro de gravidade do foguete i e o ponto de destino p;
- $angulo_i$ representa o módulo da diferença entre o ângulo atual do foguete i e o ângulo de 90 graus;
- $velocidade_i$ representa o módulo do vetor velocidade do foguete i

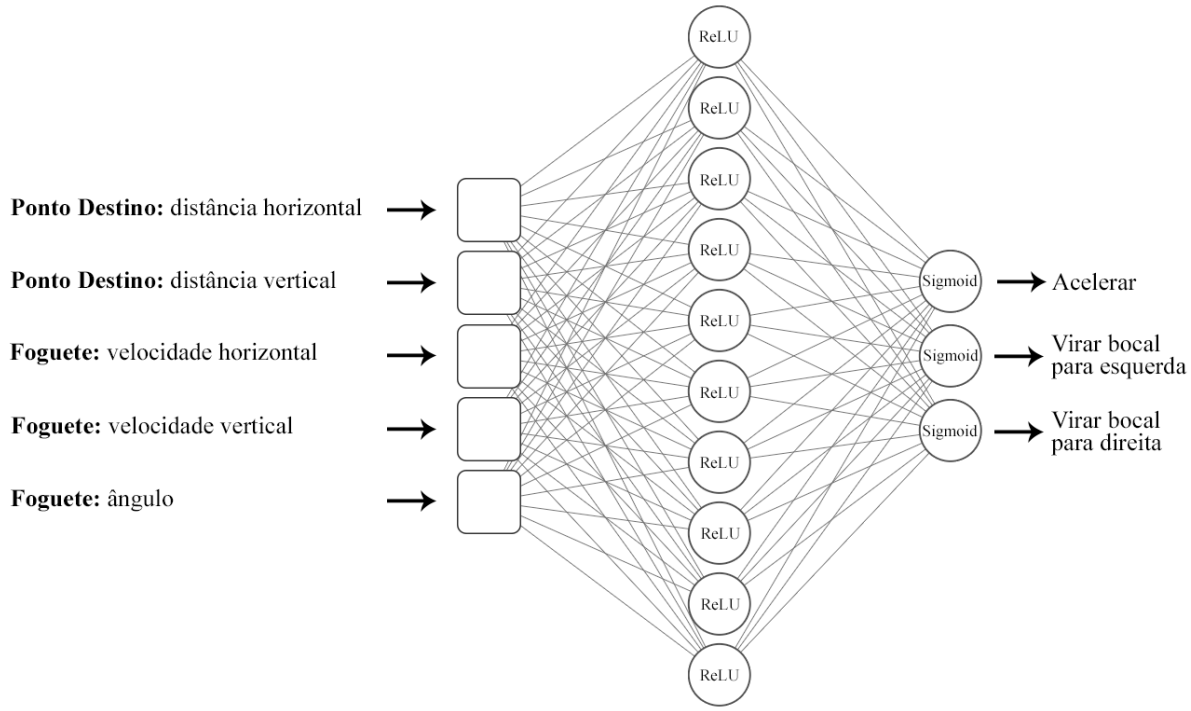
A função-aptidão será avaliada no momento em que a partida acabar. Por isso, ao final da partida, os foguetes que estiverem mais próximos do ponto destino, com um ângulo mais próximo a 90 graus e com a velocidade mais próxima de zero, serão priorizados pelo

processo evolutivo. Vale mencionar que, caso o foguete tente pousar e não consiga (pois colidiu com o navio ou com o mar), este recebe uma punição (acréscimo no valor da sua aptidão) igual à 100.

Arquitetura da Rede Neural

Considerando as cinco informações que compõem o estado do jogo e as três ações que o personagem pode executar, a arquitetura da rede neural utilizada neste jogo ficou definida como mostra a figura 3.32. Note que, os neurônios da camada de saída utilizam a função de ativação *sigmoid*. Esta função apresenta valores de saída entre 0 e 1. Estes valores serão utilizados para definir a intensidade das ações executadas.

Figura 3.32: Arquitetura da RNA utilizada no Simulador de Foguetes

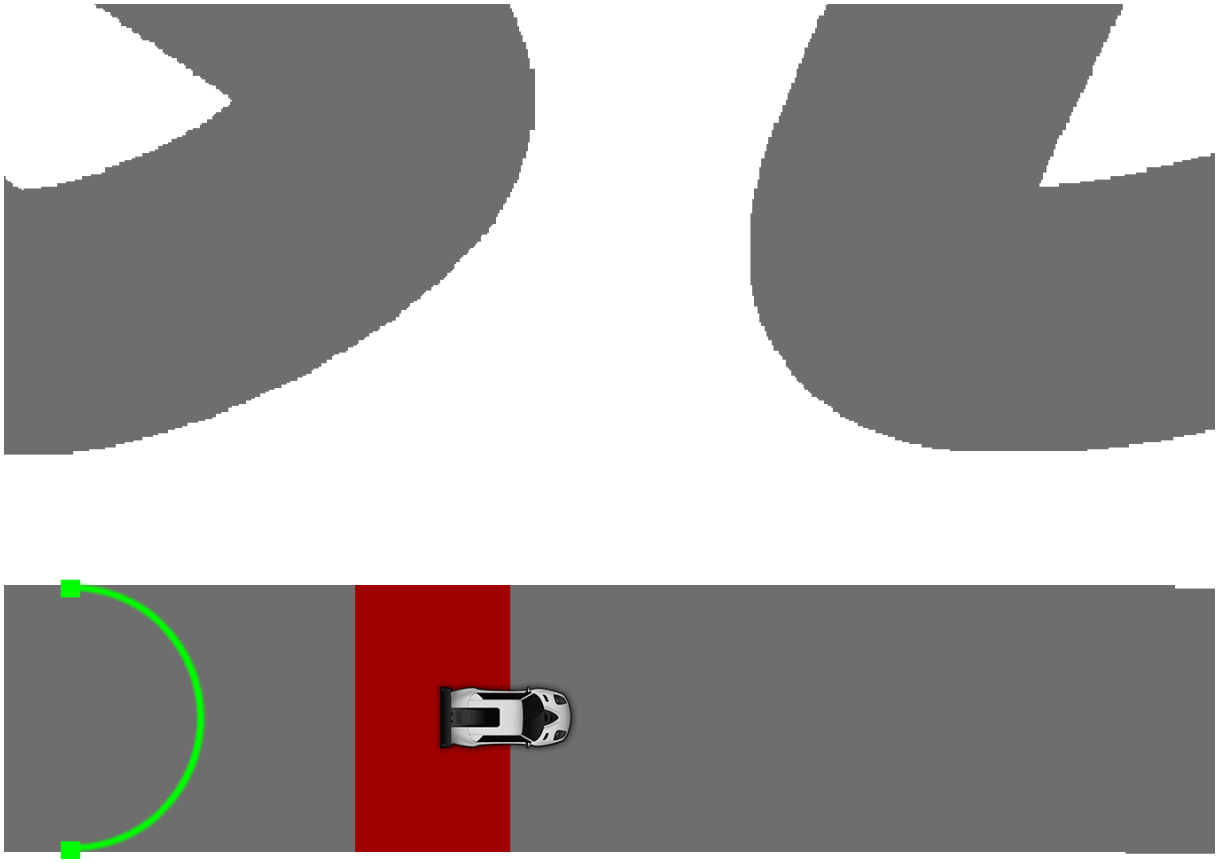


Fonte: O autor

3.4.4 Simulador de Corrida

Neste jogo, o jogador controla um carro que deve percorrer uma pista de corrida até atingir a linha de chegada. O desafio do jogo está no fato de existir um raio laser que também percorre a pista, destruindo todos os carros que entram em contato com ele. Além disso, o jogador também perde a partida caso seu carro encoste nas paredes da pista. O objetivo do jogo é atingir a linha de chegada antes que o raio laser alcance o carro. A figura 3.33 exibe uma captura de tela do jogo. Nesta figura, a região vermelha é o local onde o carro inicia a corrida.

Figura 3.33: Captura de tela do Simulador de Corrida



Fonte: O autor

Cada partida do jogo apresenta uma pista única, que é gerada em tempo real e de forma procedural. Por ser gerada proceduralmente, cada pista possui seu próprio comprimento, largura e sequência de curvas, tornando o jogo mais dinâmico. A figura 3.34 exibe alguns exemplos das pistas geradas.

Figura 3.34: Pistas geradas proceduralmente

Fonte: O autor

O processo de geração das pistas é realizado da seguinte maneira:

1. Alguns pontos são sorteados na área de um quadrado.

2. Uma rota é gerada conectando estes pontos aleatoriamente.

3. O algoritmo *2-Opt Heuristic*[42] é aplicado à rota para aprimorá-la. De forma resumida, este algoritmo realiza permutações de arestas até que nenhuma intersecção exista.

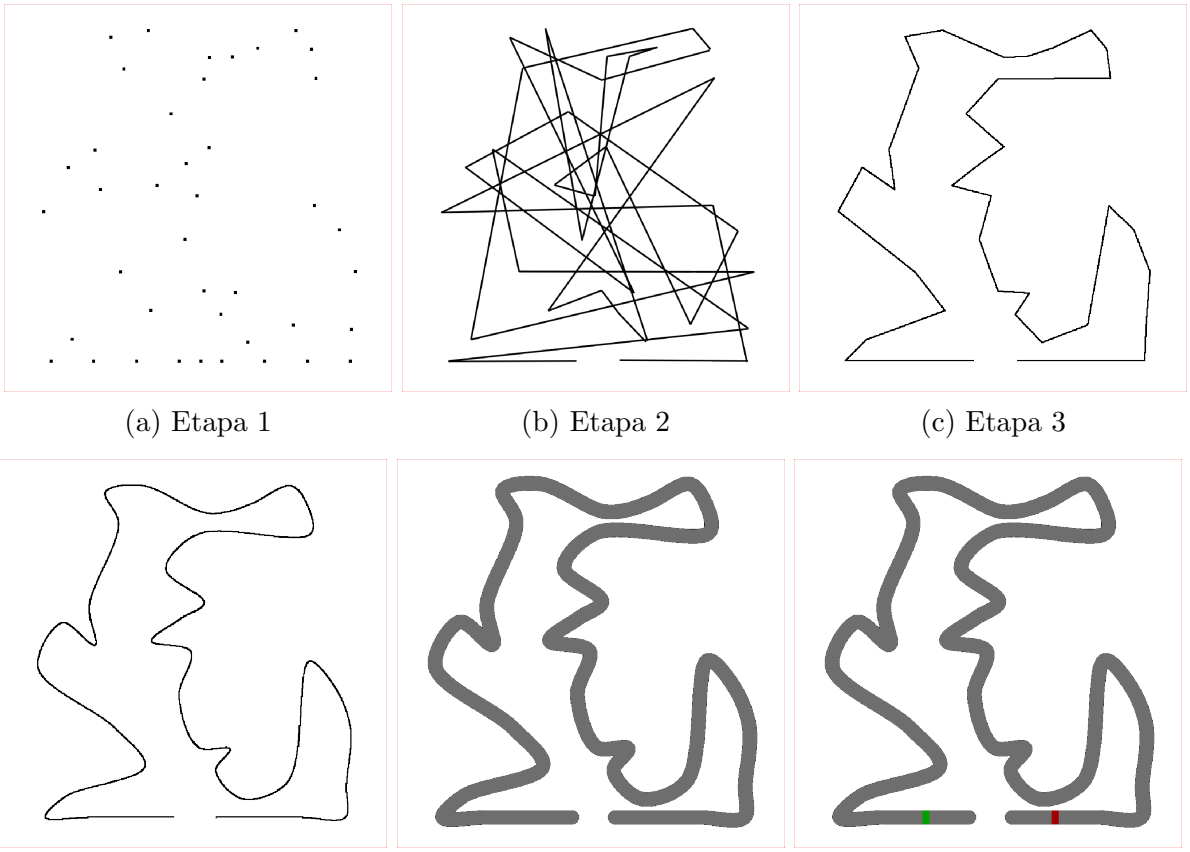
4. O algoritmo *Catmull-rom spline*[43] é aplicado à rota com o objetivo de interpolar seus pontos, diminuindo a quantidade de linhas retas e suavizando as curvas.

5. Um círculo cinza percorre a pista preenchendo a região ao redor de cada ponto pertencente à interpolação.

6. Ao final, as regiões de partida e chegada são inseridas.

A figura 3.35 exibe os estados de uma pista exemplo em cada uma dessas etapas.

Figura 3.35: Etapas do processo de geração das pistas



(a) Etapa 1 (b) Etapa 2 (c) Etapa 3

(d) Etapa 4 (e) Etapa 5 (f) Etapa 6

Fonte: O autor

Para aprimorar a pista gerada e garantir a sua validade, algumas restrições foram impostas ao processo:

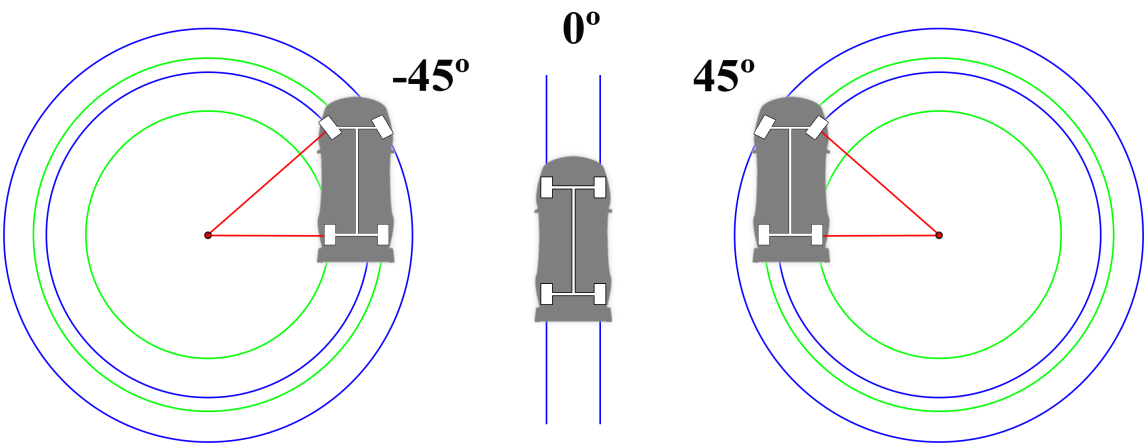
- Na etapa de sorteio dos pontos, uma distância mínima entre os pontos foi estabelecida para evitar a geração de curvas muito "fechadas" e diminuir a ocorrência de colisão entre dois trechos da pista.

- Na etapa de sorteio dos pontos, uma distância mínima das bordas do quadrado foi adotada para evitar que a interpolação ultrapasse os limites do quadrado.

- Na etapa de preenchimento, caso ocorra uma colisão entre trechos da pista, toda a pista é descartada e o processo retorna para a etapa de sorteio dos pontos.
- O formato da pista próximo às regiões de partida e chegada foi fixado em linhas retas para evitar possíveis colisões ou má formações nessas regiões.

Visando tornar o simulador mais realista, o movimento do carro foi implementado de forma parecida com o movimento de um carro real. O eixo perpendicular à roda dianteira em conjunto com o eixo perpendicular à roda traseira possuem um ponto de intersecção (exceto quando estes eixos são paralelos entre si). Este ponto determina o centro das circunferências que as rodas do carro percorrerão. Caso os eixos perpendiculares às rodas sejam paralelos entre si, as rodas percorrerão linhas retas. A figura 3.36 exibe um esquema que exemplifica esse modelo de movimento. Na figura, as linhas azuis representam as trajetórias das rodas dianteiras e as linhas verdes as trajetórias das rodas traseiras. Os ângulos exibidos na figura são os ângulos do volante do carro em cada uma das três situações.

Figura 3.36: Modelo de movimento do carro



Fonte: O autor

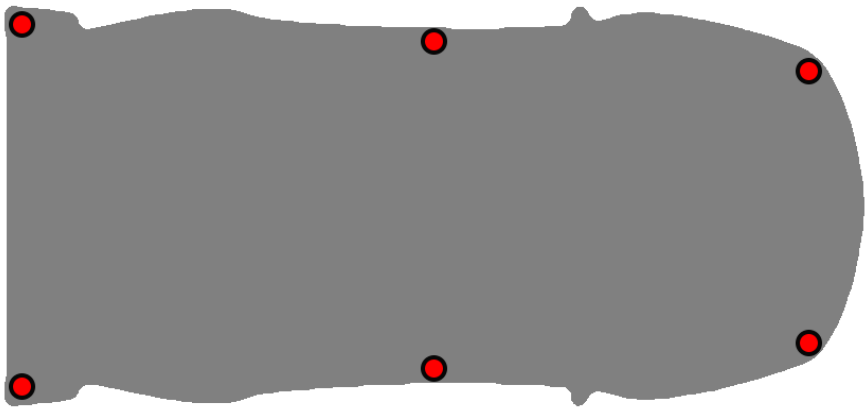
Caso o carro esteja "à deriva" (isto é, nem acelerando, nem freando), uma taxa de atrito é aplicada ao carro, para que este perca lentamente velocidade até que pare de se movimentar.

De acordo com este modelo de movimento, o jogador dispõe de quatro ações para controlar o carro: acelerar, frear, girar volante para a esquerda e girar volante para a direita. Ao acelerar, uma determinada quantidade a é adicionada à velocidade angular do

carro. Ao frear, a velocidade do carro é multiplicada por uma constante no intervalo $[0, 1]$. Diante disso, pode-se perceber que a aceleração aumenta a velocidade do carro de forma linear enquanto o freio diminui a velocidade do carro de forma exponencial. Ao girar o volante, um incremento (ou decremento) é adicionado ao ângulo das rodas dianteiras. O ângulo máximo que as rodas podem atingir é de 45 graus em relação ao carro. Além disso, vale destacar que os carros não possuem marcha ré.

Para detectar a colisão do carro com as paredes da pista ou com o laser, foram utilizados seis pontos específicos ao redor do carro. Quando um destes pontos encosta no laser ou sai da pista, o carro é considerado "colidido". A figura 3.37 exibe esses pontos.

Figura 3.37: Pontos de colisão do carro



Fonte: O autor

A tabela 3.4 apresenta os valores de alguns parâmetros utilizados na implementação do jogo.

Tabela 3.4: Parâmetros do Simulador de Corrida

velocidadeMaximaCarro: 500	anguloMaximoVolante: 45
velocidadeLaser: 0.5	anguloMinimoVolante: -45
penalidadeColisaoParede: 1	comprimentoCarro: 40
taxaDeAtrito: 0.99	larguraCarro: 20
taxadeFreioCarro: 0.97	aceleracaoCarro: 1
larguraMaximaPista: 100	velocidadeGiroVolante: 2
larguraMinimaPista: 40	alcanceMaximoSensor: 500

Por fim, vale ressaltar que o raio laser inicia a partida posicionado no primeiro ponto da pista (antes da linha de partida dos carros) e se movimenta com velocidade constante ao longo da pista.

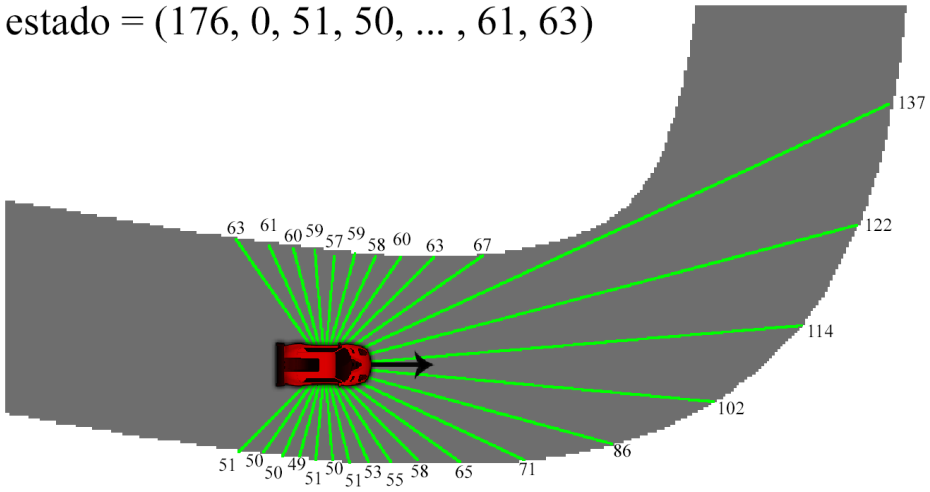
Modelagem do Estado

Neste jogo, o vetor de estados foi modelado utilizando as seguintes informações:

- A velocidade angular do carro.
- O ângulo do volante do carro.
- 27 sensores de distância distribuídos igualmente ao longo de 270º. Para que o formato da pista esteja presente no vetor que representa o estado do jogo, foram adicionados 27 sensores de distância ao redor do carro. Cada sensor dispara um feixe de luz que percorre uma linha reta até colidir com a parede ou alcançar o comprimento máximo de 500 *pixels*. O comprimento do feixe é o valor utilizado no vetor de estados. A implementação desses sensores visa simular o funcionamento de um LIDAR[44], um sistema real que mapeia a distância entre o carro e os obstáculos do ambiente através de pulsos de luz.

A figura 3.38 exibe um exemplo da modelagem do estado.

Figura 3.38: Modelo do estado do simulador de corrida

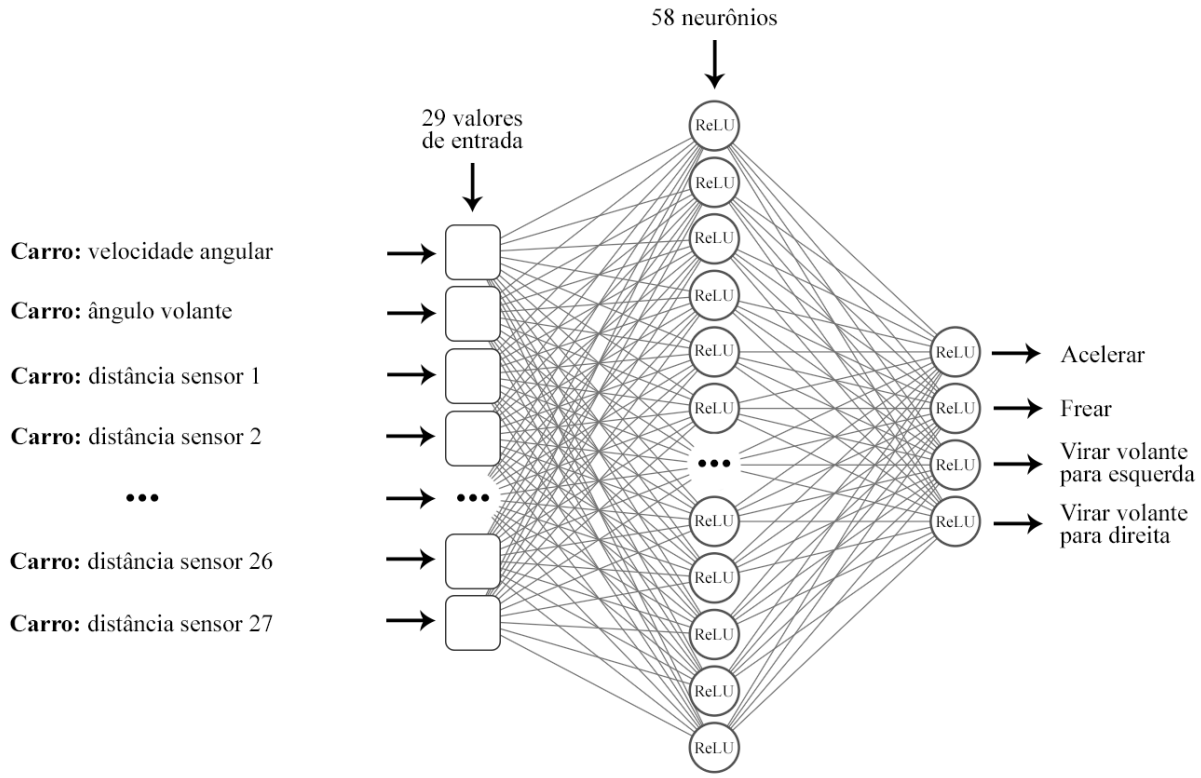


Fonte: O autor

Arquitetura da Rede Neural

Considerando as 29 informações (velocidade do carro, ângulo do volante e 27 sensores de distância) que compõem o estado do jogo e as quatro ações que o carro pode executar, a arquitetura da rede neural utilizada neste jogo ficou definida como mostra a figura 3.39.

Figura 3.39: Arquitetura da RNA utilizada no simulador de corrida



Fonte: O autor

Função-Aptidão

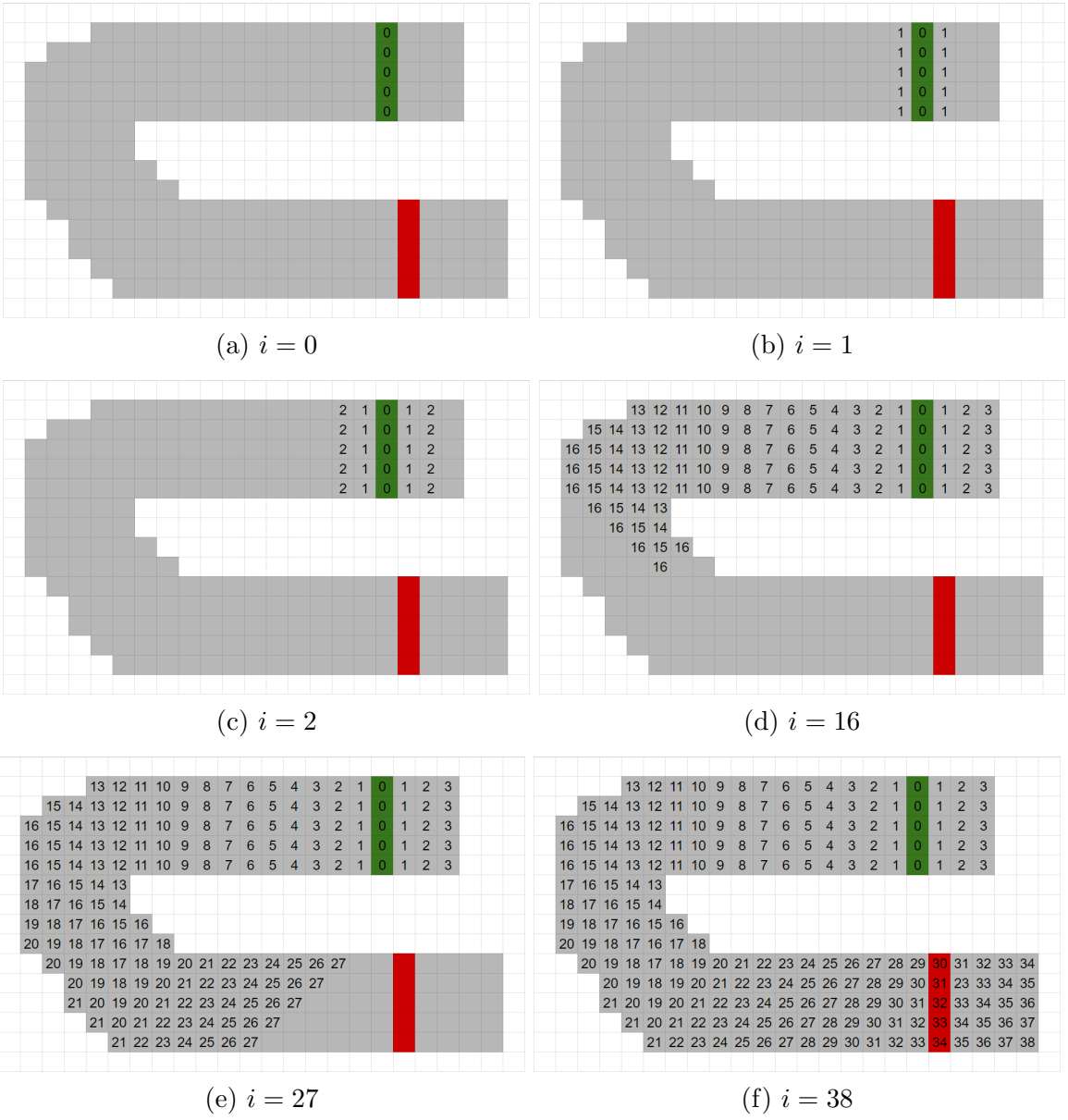
Neste jogo, a função-aptidão que deve ser minimizada foi definida da seguinte maneira:

$$f(i) = distanciaRestante_i$$

onde $distanciaRestante_i$ representa a distância entre o indivíduo i e a linha de chegada.

Para calcular esta distância, foi utilizado o *Wavefront expansion algorithm*[45]. Este algoritmo utiliza uma matriz para armazenar a distância de todos os pontos do cenário até os pontos de destino (neste caso, a linha de chegada). Para preencher essa matriz, primeiramente, o algoritmo inicializa os pontos de destino com o valor zero. Em seguida, um *loop* é iniciado atualizando as distâncias de todos os outros pontos da matriz. Essa atualização é feita de forma que os pontos recebam a distância do seu menor vizinho somada de uma unidade. O *loop* permanece em execução até que todos os pontos tenham sido avaliados. A figura 3.40 demonstra o funcionamento do algoritmo exibindo o estado da matriz de distâncias ao longo de algumas iterações i.

Figura 3.40: Execução do *Wavefront expansion algorithm*



(a) $i = 0$ (b) $i = 1$

(c) $i = 2$ (d) $i = 16$

(e) $i = 27$ (f) $i = 38$

Fonte: O autor

No entanto, vale ressaltar que, ao utilizar as distâncias absolutas das pistas como aptidão, o resultado do processo evolutivo pode ser impactado. Pois pistas menores possuem valores de aptidão menores. Dessa forma, o processo pode dar prioridade para as pistas pequenas e enviesar os indivíduos, fazendo-os aprender a dirigir apenas nas pistas pequenas. Para que o comprimento da pista não atrapalhe o processo evolutivo, o valor da função-aptidão foi normalizado. Assim, todas as pistas apresentarão valores de aptidão entre 0 e 100. Além disso, uma penalidade de valor igual à 1 é aplicada ao indivíduo caso ele perca o jogo colidindo com uma parede. Essa penalidade tem o objetivo de desincentivar o comportamento dos indivíduos de colidirem com paredes.

3.5 Interface Gráfica

Para facilitar a visualização e a interação com as etapas de treinamento e teste das redes neurais, foi desenvolvida uma interface gráfica utilizando a mesma biblioteca gráfica aplicada aos jogos. A figura 3.41 exibe esta interface. A coluna mais à direita exibe informações em tempo real a respeito do indivíduo com a melhor aptidão que ainda está "vivo" e jogando. A região central (cinza) é reservada para a exibição do jogo em questão.

Figura 3.41: Interface gráfica: Treinamento

Fonte: O autor

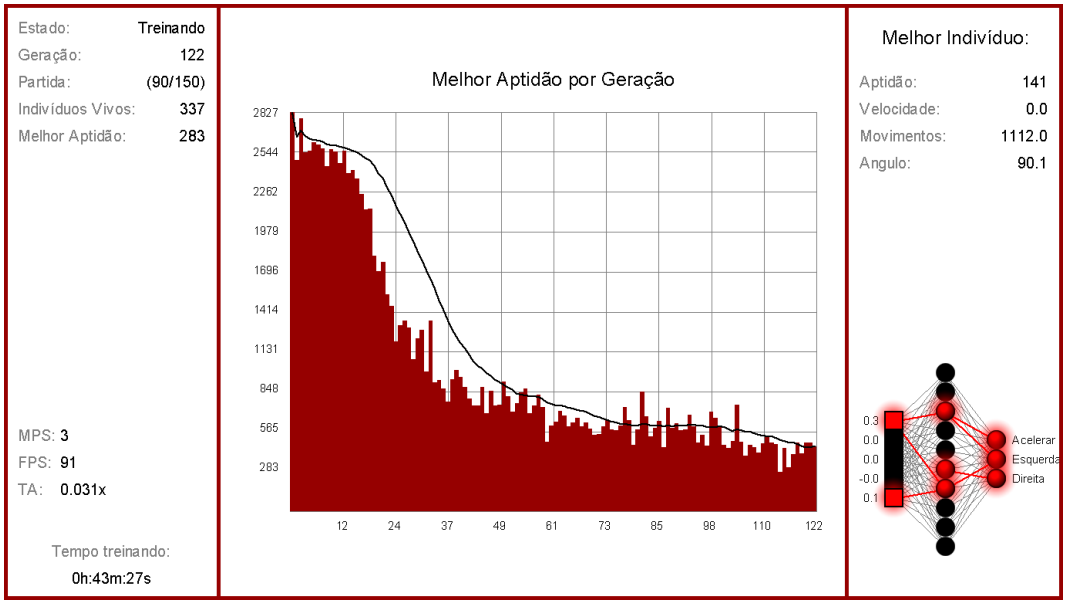
Durante o treinamento, a coluna mais à esquerda exibe informações gerais sobre o treinamento e um gráfico com a evolução da aptidão, como mostra a figura 3.41. No entanto, durante a etapa de validação, esta coluna tem seu comportamento alterado e passa a exibir informações gerais sobre o indivíduo que está sendo validado, além de um gráfico com a performance deste indivíduo em cada partida de validação. A figura 3.42 exibe isto.

Figura 3.42: Interface gráfica: Validação

Fonte: O autor

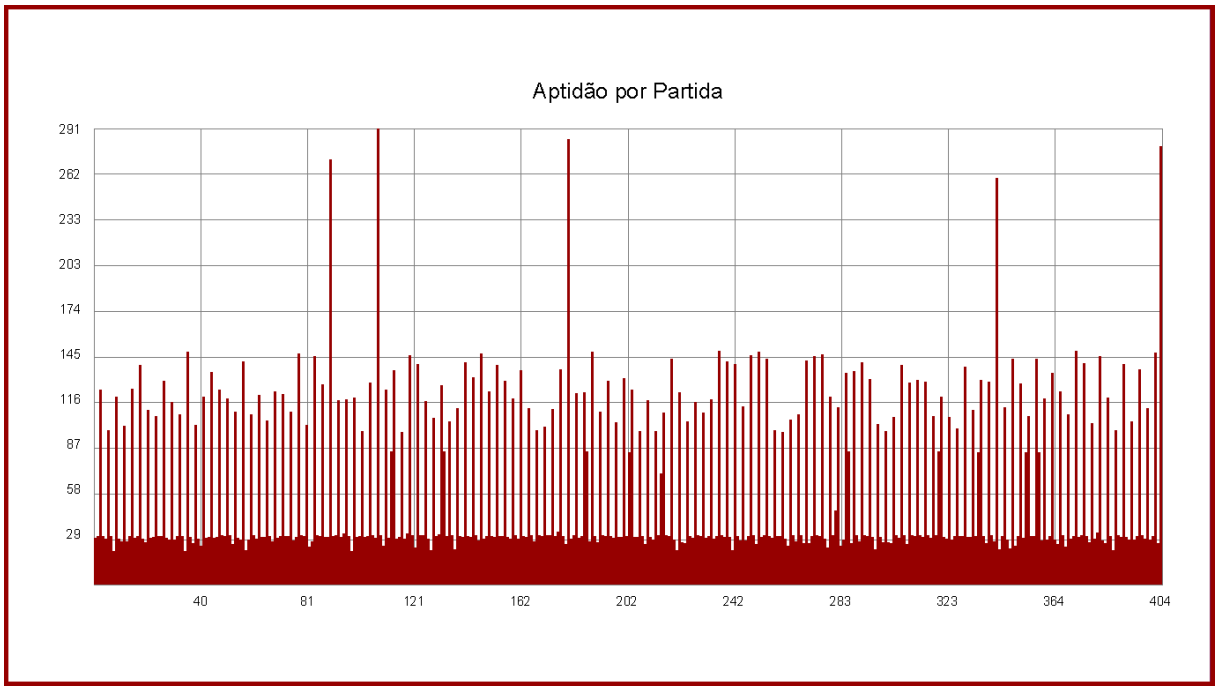
Além de fornecer informações, a interface gráfica também permite ampliar os gráficos e o esquema que representa a rede neural, como mostram as figuras 3.43, 3.44 e 3.45.

Figura 3.43: Interface gráfica: Gráfico Ampliado



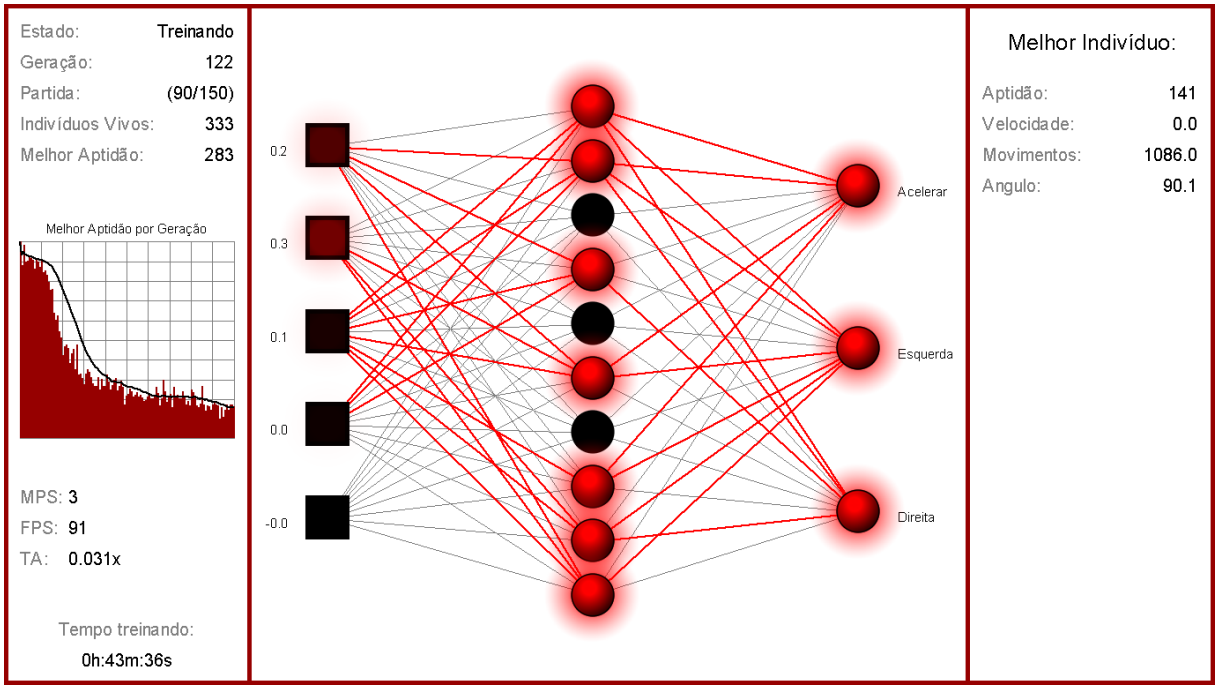
Fonte: O autor

Figura 3.44: Interface gráfica: Gráfico Ampliado e Expandido



Fonte: O autor

Figura 3.45: Interface gráfica: Rede Neural Ampliada



Fonte: O autor

Capítulo 4

Resultados

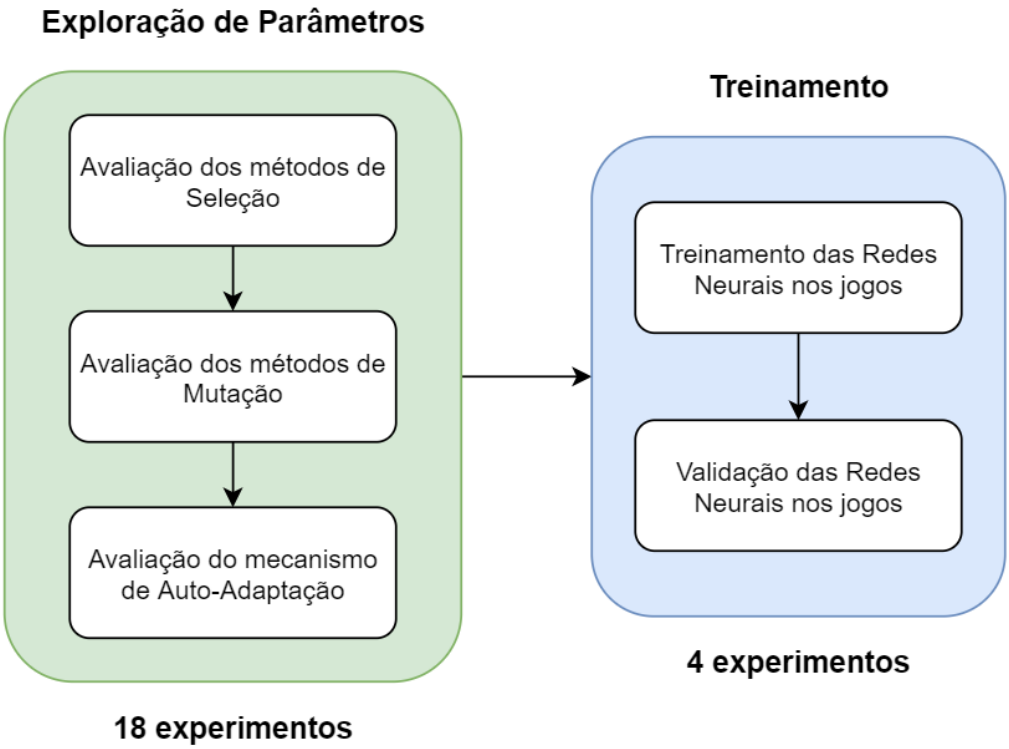
Para atingir o objetivo do trabalho, foram realizados alguns experimentos e os resultados serão apresentados neste capítulo. Neste trabalho, o termo "experimento" representa um conjunto com um ou mais treinamentos, realizados com o objetivo de coletar informações sobre o desempenho da EE em uma determinada situação. Ao todo, foram realizados 22 experimentos com 24 horas de duração cada, totalizando 528 horas de execução. Estes experimentos foram organizados e executados da seguinte maneira:

1. Exploração de parâmetros: Nesta etapa, foram realizados 18 experimentos para avaliar o desempenho dos diferentes métodos de seleção/mutação e a efetividade do mecanismo de Auto-Adaptação. Cada experimento foi composto por 12 treinamentos com 2 horas de duração cada. A finalidade desta etapa foi a de encontrar os melhores valores para os parâmetros da EE no contexto deste trabalho.

2. Treinamento: Após determinada a melhor configuração para a EE, esta foi utilizada para, de fato, treinar as redes neurais nos jogos implementados. Estes experimentos foram compostos por apenas um treinamento com 24 horas de duração. A finalidade desta etapa foi a de obter redes neurais que possuam um elevado desempenho nos jogos propostos.

A figura 4.1 exibe um esquema que ilustra cada grupo de experimentos e a ordem em que foram realizados:

Figura 4.1: Experimentos realizados



Fonte: O autor

Todos os experimentos foram realizados em uma máquina com as seguintes características:

- Sistema Operacional: Windows 10;
- Processador: AMD Ryzen 7 2700, Eight-Core, 3.20 GHz;
- Memória Principal: 16 GB.

Vale ressaltar que nenhuma GPU foi utilizada para o processamento de informações. No entanto, o processo de *feedforward* realizado pelas redes neurais e o processamento dos personagens do jogo foram paralelizados na CPU utilizando 10 *threads*. Além disso, durante o treinamento, o desenho e a atualização dos elementos gráficos da tela foram interrompidos. Dessa forma, uma parte considerável de tempo de CPU foi poupada e pôde ser redirecionada para o processamento da lógica do jogo e das redes neurais. Vale mencionar também que a velocidade de processamento do jogo foi configurada para operar em sua capacidade máxima durante os treinamentos.

Por fim, a quantidade de progenitores (μ) e a quantidade descendentes (λ) utilizada nas populações da estratégia evolutiva permaneceram constantes em todos os experimentos e foram definidas da seguinte maneira:

- *Flappy Bird*, Dinossauro do *Google Chrome* e Simulador de Foguete: $\mu = 100$ e $\lambda = 1000$
- Simulador de Corrida: $\mu = 50$ e $\lambda = 500$

Em razão do custo computacional de se calcular os sensores de distância dos carros no Simulador de Corrida, este teve a quantidade de indivíduos das populações reduzida pela metade em comparação aos outros jogos.

Por fim, vale ressaltar que os jogos *Flappy Bird* e Dinossauro do *Google Chrome* possuem uma aptidão crescente (ou seja, quanto maior, melhor) enquanto os jogos Simulador de Foguetes e Simulador de Corrida possuem aptidão decrescente (quanto menor, melhor).

4.1 Exploração de parâmetros

Para avaliar o impacto de cada parâmetro da estratégia evolutiva na efetividade do treinamento das redes neurais, foram realizados 12 treinamentos para cada valor proposto de cada parâmetro avaliado. Cada treinamento possuiu duas horas de duração, totalizando assim, 24 horas por experimento. A razão pela qual foram realizados vários treinamentos, foi a de minimizar os efeitos da aleatoriedade do processo evolutivo nos resultados da avaliação. De modo geral, a exploração dos parâmetros foi realizada da seguinte maneira:

1. Primeiramente, os dois métodos de seleção propostos por Rechenberg[20] foram avaliados e comparados. O método que apresentou o melhor resultado foi o método escolhido para ser utilizado nos próximos experimentos.
2. Após definir o melhor método de seleção, dois métodos de mutação foram avaliados e comparados: O método proposto por Rechenberg e o método proposto por este trabalho, denominado “Mutações Parciais”. O método que apresentou os melhores resultados também foi o método escolhido para ser utilizado nos experimentos seguintes.

3. Após definidos os métodos de seleção e mutação, mais um experimento foi realizado, porém, desta vez, com o mecanismo de Auto-Adaptação ativado.

Por fim, após definidos os melhores valores para cada parâmetro, estes foram usados na segunda etapa dos experimentos.

4.1.1 Experimentando métodos de seleção

Nesta subseção, serão apresentados os resultados da avaliação e da comparação dos métodos de seleção (μ, λ) e $(\mu + \lambda)$ propostos por Rechenberg[20]. Nestes experimentos, a quantidade de partidas que os indivíduos jogaram antes de terem suas aptidões calculadas foi definida da seguinte forma:

- *Flappy Bird*, Dinossauro do *Google Chrome* e Simulador de Corrida: 25 partidas
- Simulador de Foguete: 300 partidas (sendo 100 partidas para decolar, 100 para voar e 100 para pousar).

Em razão da pequena duração das partidas do Simulador de Foguetes, este pôde ter a quantidade de partidas aumentada em relação aos outros jogos, tornando sua função-aptidão mais robusta.

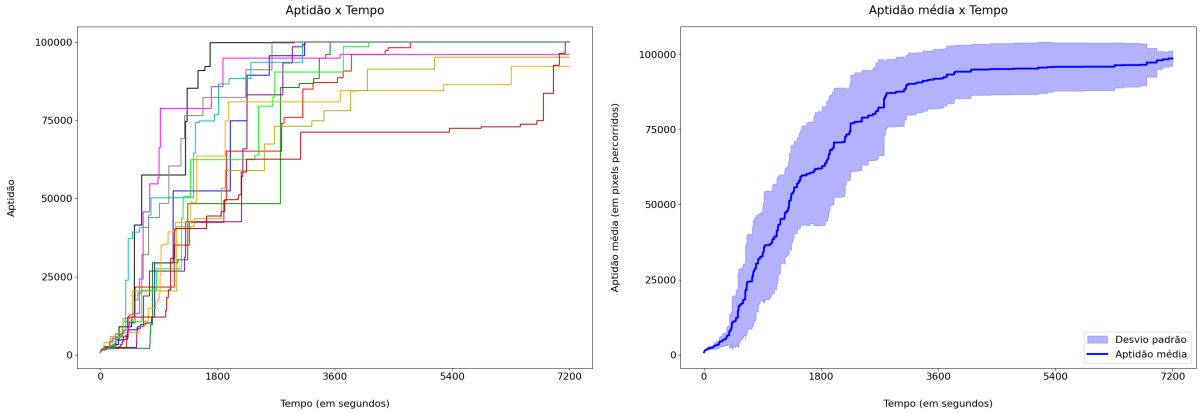
Os parâmetros da estratégia evolutiva que não foram avaliados nesta etapa permaneceram fixos e com os seguintes valores:

- Método de Mutação: método original proposto por Rechenberg[20]
- Valor do desvio padrão do operador de mutação: 1
- Auto-Adaptação: desligada.

Experimentando o método de seleção (μ, λ)

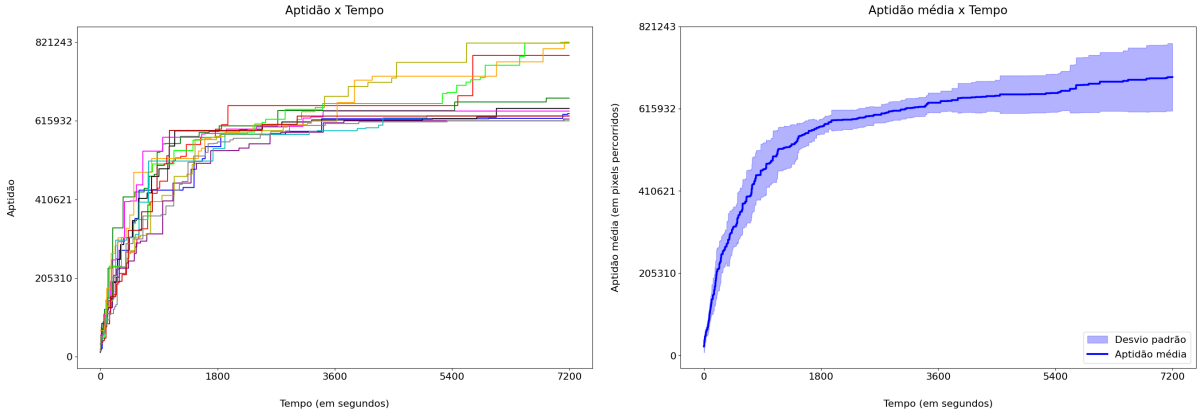
As figuras 4.2, 4.3, 4.4 e 4.5 exibem os resultados da aplicação do método de seleção (μ, λ) no treino das redes neurais em cada um dos jogos implementados. O gráfico à esquerda exibe a aptidão do melhor indivíduo ao longo do tempo. Cada linha deste gráfico representa um treinamento distinto. O gráfico à direita foi obtido através do cálculo da média e do desvio padrão de todas as linhas do gráfico à esquerda.

Figura 4.2: Desempenho do método de seleção (μ, λ) aplicado ao Flappy Bird



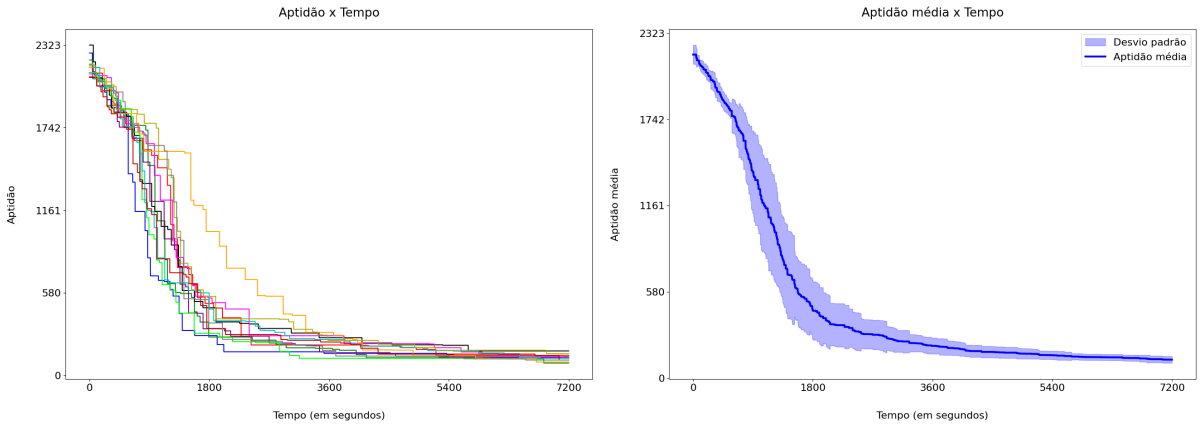
Fonte: O autor

Figura 4.3: Desempenho do método de seleção (μ, λ) aplicado ao Dinossauro do Google Chrome



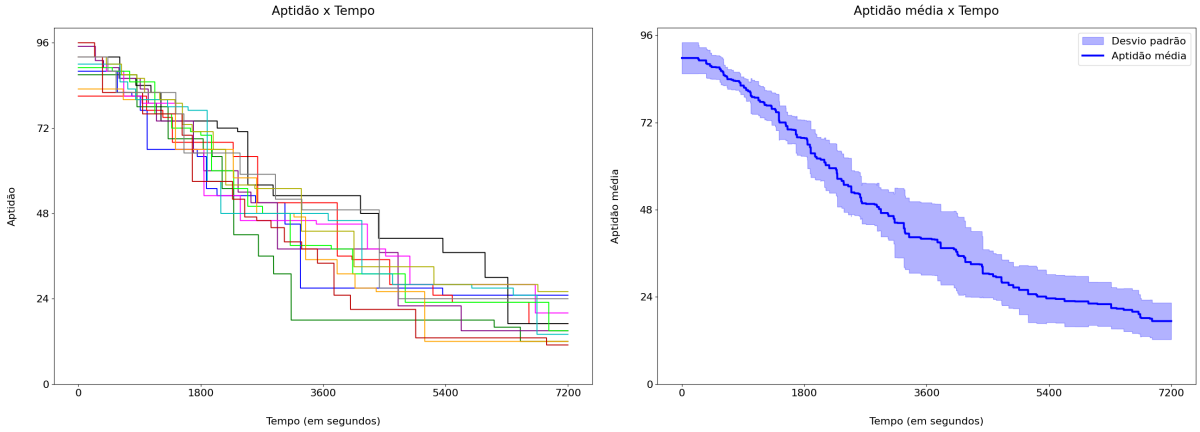
Fonte: O autor

Figura 4.4: Desempenho do método de seleção (μ, λ) aplicado ao Simulador de Foguetes



Fonte: O autor

Figura 4.5: Desempenho do método de seleção (μ, λ) aplicado ao Simulador de Corrida

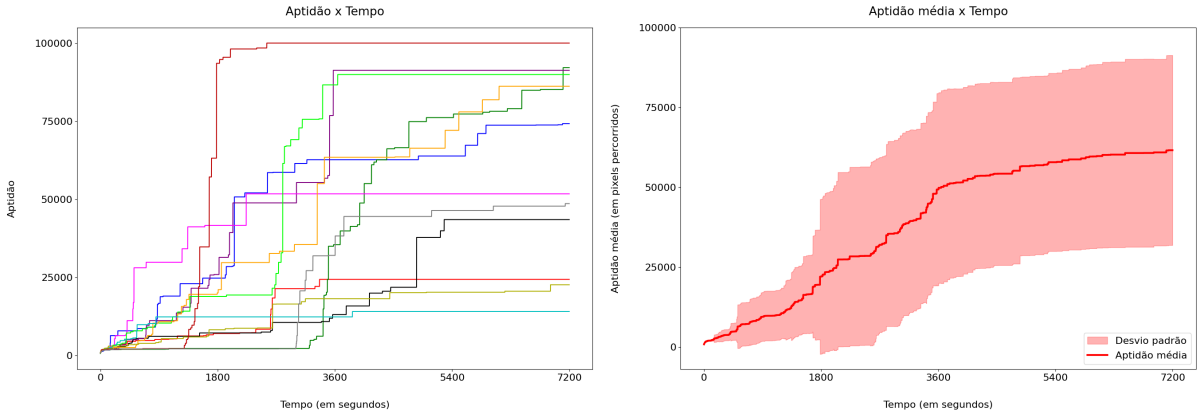


Fonte: O autor

Experimentando o método de seleção $(\mu + \lambda)$

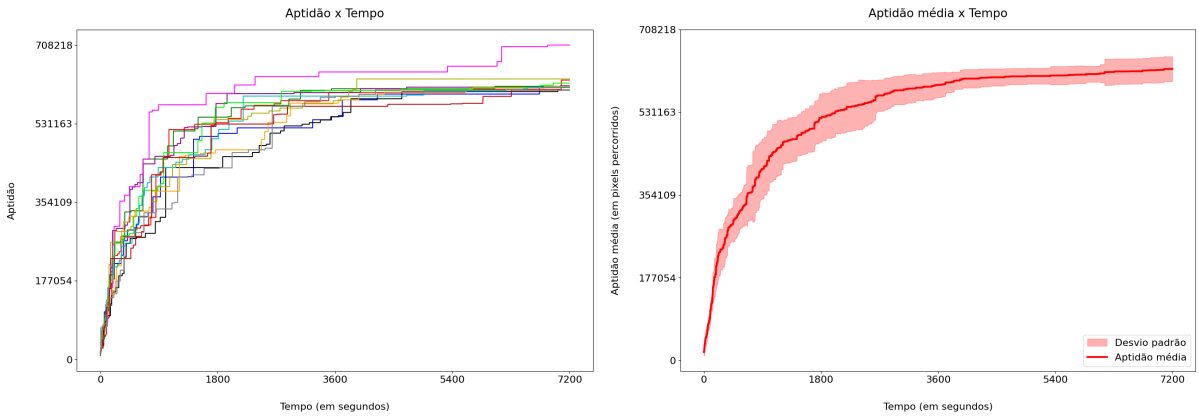
As figuras 4.6, 4.7, 4.8 e 4.9 exibem os resultados da aplicação do método de seleção (μ, λ) no treino das redes neurais em cada um dos jogos implementados. Vale destacar que, de acordo com a figura 4.6, muitos treinamentos não conseguiram encontrar um indivíduo que atingisse a aptidão média máxima (100.000 pontos) no jogo Flappy Bird.

Figura 4.6: Desempenho do método de seleção $(\mu + \lambda)$ aplicado ao Flappy Bird



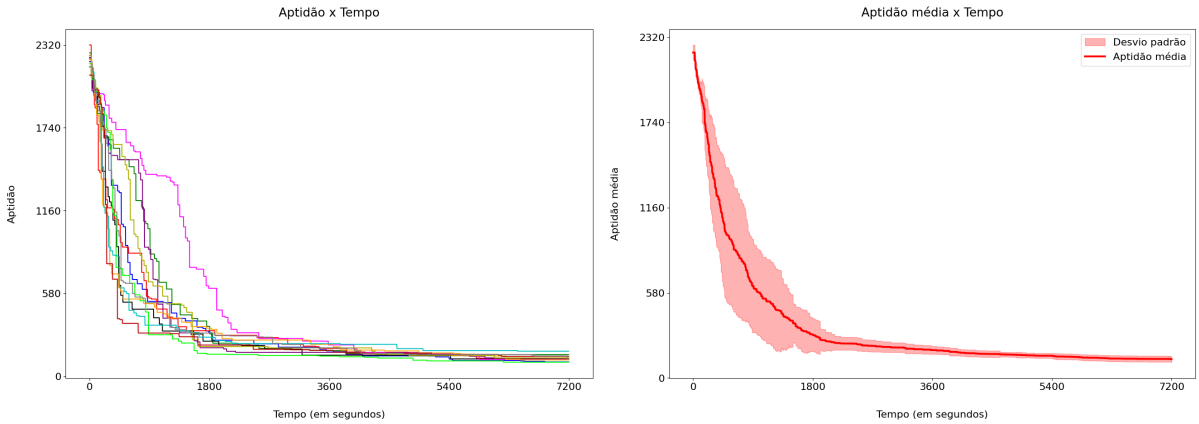
Fonte: O autor

Figura 4.7: Desempenho do método de seleção $(\mu + \lambda)$ aplicado ao Dinossauro do Google Chrome



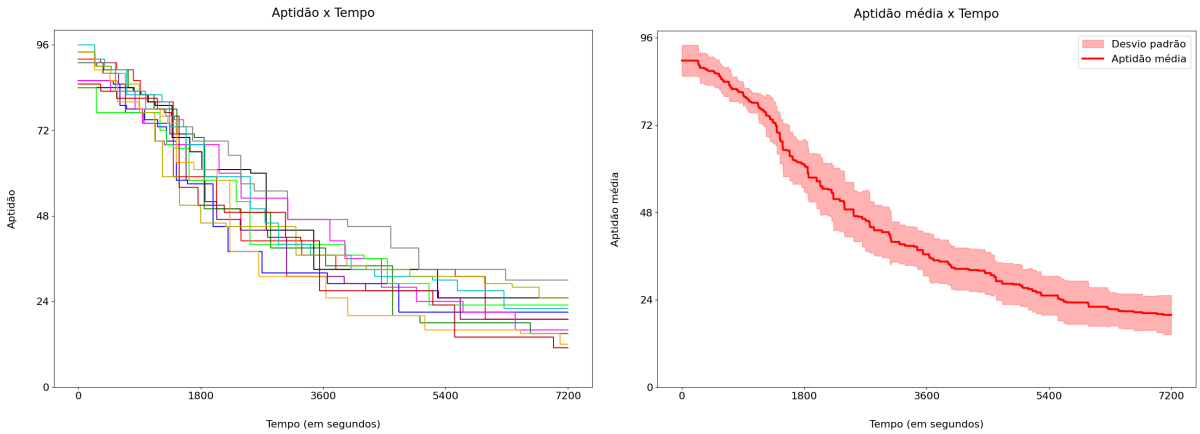
Fonte: O autor

Figura 4.8: Desempenho do método de seleção $(\mu + \lambda)$ aplicado ao Simulador de Foguetes



Fonte: O autor

Figura 4.9: Desempenho do método de seleção $(\mu + \lambda)$ aplicado ao Simulador de Corrida

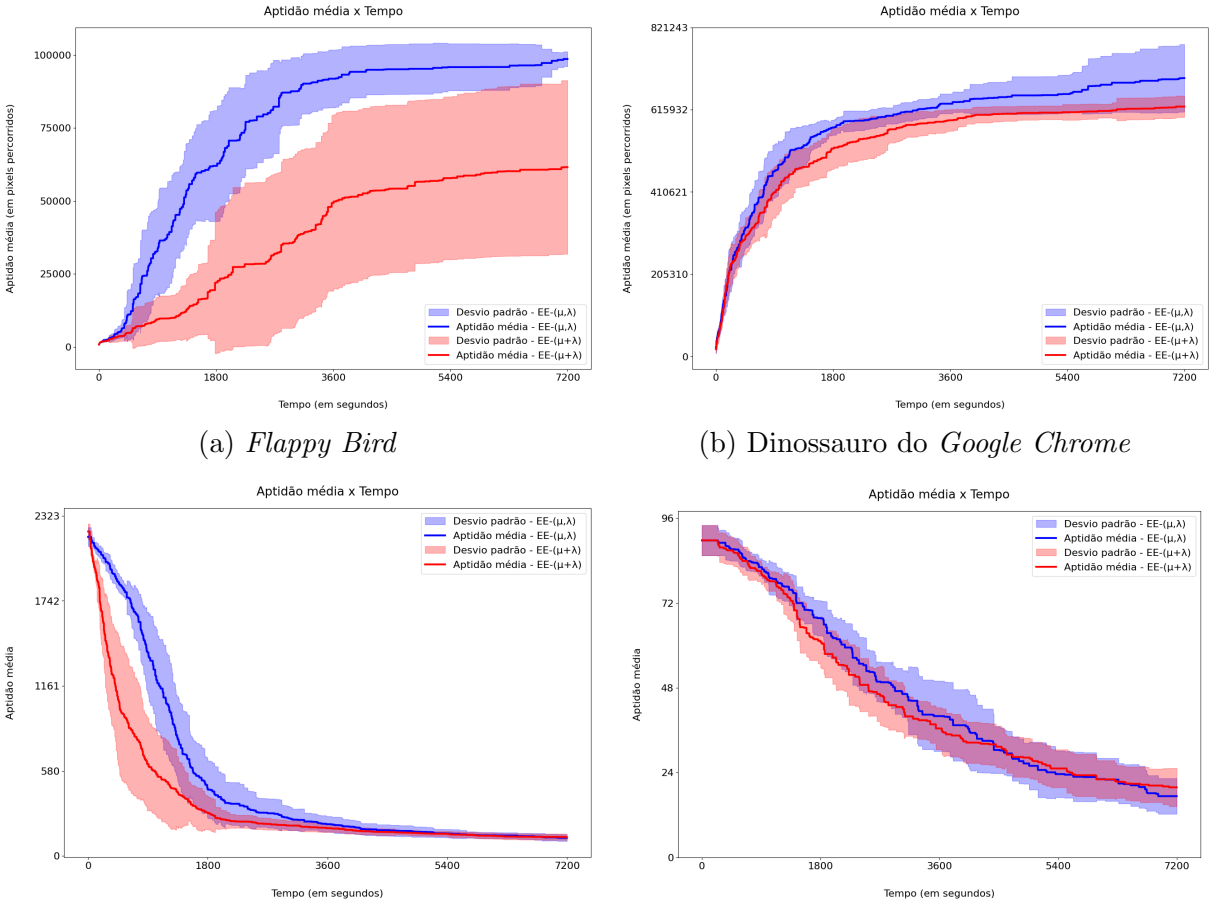


Fonte: O autor

Comparação entre os métodos (μ, λ) e $(\mu + \lambda)$

Para identificar qual método apresentou o melhor desempenho nos jogos implementados, os gráficos exibidos nas subseções anteriores foram sobrepostos e comparados. A figura 4.10 exibe esta sobreposição para cada um dos jogos propostos.

Figura 4.10: Comparação entre o desempenho dos métodos (μ, λ) e $(\mu + \lambda)$



(a) *Flappy Bird* (b) Dinossauro do *Google Chrome*

(c) Simulador de Foguetes (d) Simulador de Corrida

Fonte: O autor

Analisando os gráficos exibidos na figura 4.10, pode-se notar que:

- De acordo com a figura 4.10a, o método (μ, λ) apresentou um desempenho significativamente superior ao método $(\mu + \lambda)$ no jogo *Flappy Bird*, uma vez que esse método demonstrou uma aptidão média maior e um desvio padrão menor durante todo o treinamento.

- Pela figura 4.10b, o método (μ, λ) também apresentou um desempenho superior ao método $(\mu + \lambda)$ no jogo do Dinossauro do *Google Chrome*. Porém, neste jogo, a diferença entre os métodos foi consideravelmente menor. Ao final do treinamento,

pode-se notar que o método (μ, λ) demonstra maior capacidade de escapar de ótimos locais e continuar o processo de otimização, enquanto o método $(\mu + \lambda)$ demonstra estagnação.

- De acordo com a figura 4.10c, o método $(\mu + \lambda)$ demonstrou maior velocidade de convergência em comparação com o método (μ, λ) no Simulador de Foguetes. Porém, ao final do treinamento, ambos atingiram os mesmos valores de aptidão média.

- Pela figura 4.10d, o método $(\mu+\lambda)$ demonstrou uma convergência ligeiramente mais veloz. Porém, ao final do treinamento, o método (μ, λ) apresentou um resultado ligeiramente melhor.

Portanto, de acordo com os dados apresentados, o método (μ, λ) demonstrou um desempenho igual ou superior ao método $(\mu + \lambda)$ nos jogos propostos. Por este motivo, o método (μ, λ) foi escolhido para ser o método de seleção utilizado nos próximos experimentos.

4.1.2 Experimentando métodos de mutação

Nesta subseção, serão apresentados os resultados da avaliação e da comparação entre o método de mutação proposto por Rechenberg[20] e o método proposto por este trabalho. Cada um dos métodos possui seu próprio parâmetro que determina a intensidade da mutação. No caso da mutação de Rechenberg, este parâmetro é um valor real que define a magnitude dos ruídos aleatórios adicionados aos genes dos indivíduos. Este parâmetro é denominado "desvio padrão" e será representado pelo símbolo D. Por outro lado, a mutação proposta neste trabalho, apresenta um parâmetro m composto por um valor inteiro. Este valor determina a amplitude do sorteio da quantidade de mutações que o indivíduo receberá.

Por terem naturezas diferentes e representarem técnicas diferentes, realizar apenas um experimento para cada método de mutação (com $D = m$) pode não ser uma comparação muito adequada. Pois um valor que pode ser ideal para o parâmetro D, pode não ser para o parâmetro m (por exemplo, o valor 1 pode ser bom para D, porém, é demasiadamente pequeno para m). Por este motivo, diferentes valores para D e m foram experimentados:

- Mutação de Rechenberg (D): 0.1, 1, 10
- Mutações Parciais (m): 2, 4, 8, 16, 32, 64

No entanto, após avaliados, apenas o valor que apresentou o melhor desempenho para D foi comparado ao valor que apresentou o melhor desempenho para m.

Para que a execução dessa grande quantidade de experimentos fosse viável (em questão de tempo), estes foram realizados em apenas um jogo. O jogo escolhido para serem realizados os experimentos foi o Simulador de Corrida. Este jogo foi escolhido pois, de acordo com a figura 4.10, demonstrou ser o mais desafiador para as redes neurais, visto que não apresentou uma convergência definitiva nos experimentos anteriores.

Para viabilizar que a convergência aconteça dentro do intervalo de duas horas do treinamento, a quantidade de partidas que os indivíduos jogaram antes de terem suas aptidões calculadas foi reduzida de 25 para 10.

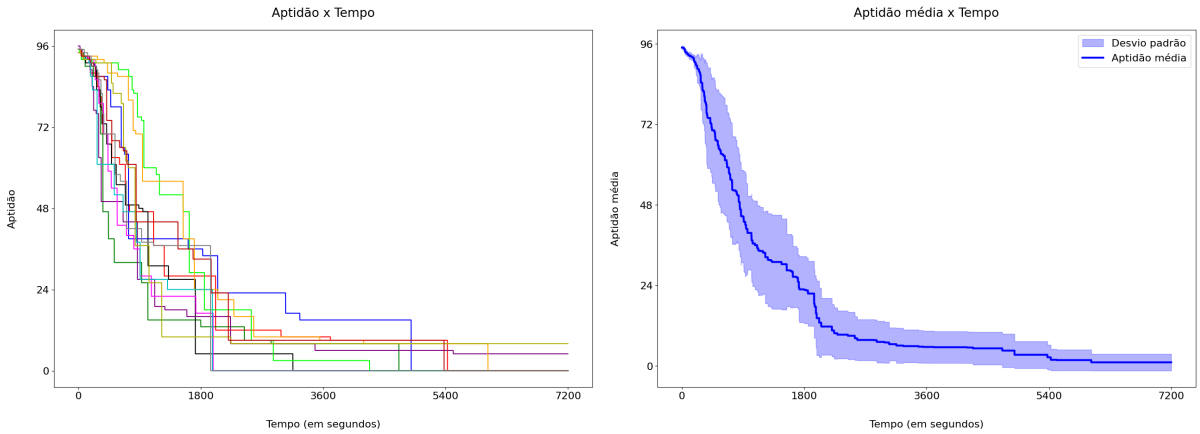
Os parâmetros da estratégia evolutiva que não foram avaliados nesta etapa permaneceram fixos e com os seguintes valores:

- Método de Seleção: (μ, λ)
- Auto-Adaptação: desligada.

Experimentando a mutação de Rechenberg

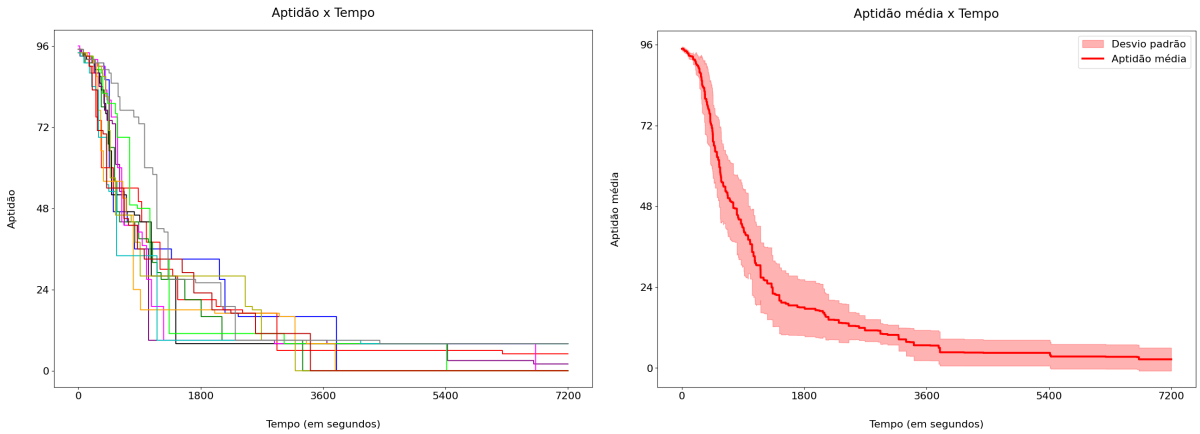
As figuras 4.11, 4.12 e 4.13 exibem os resultados da aplicação da mutação de Rechenberg no Simulador de Corrida. Nestes experimentos, o valor do parâmetro D assumiu os seguintes valores: 0.1, 1 e 10.

Figura 4.11: Desempenho da mutação de Rechenberg no Simulador de Corrida ($D = 0.1$)



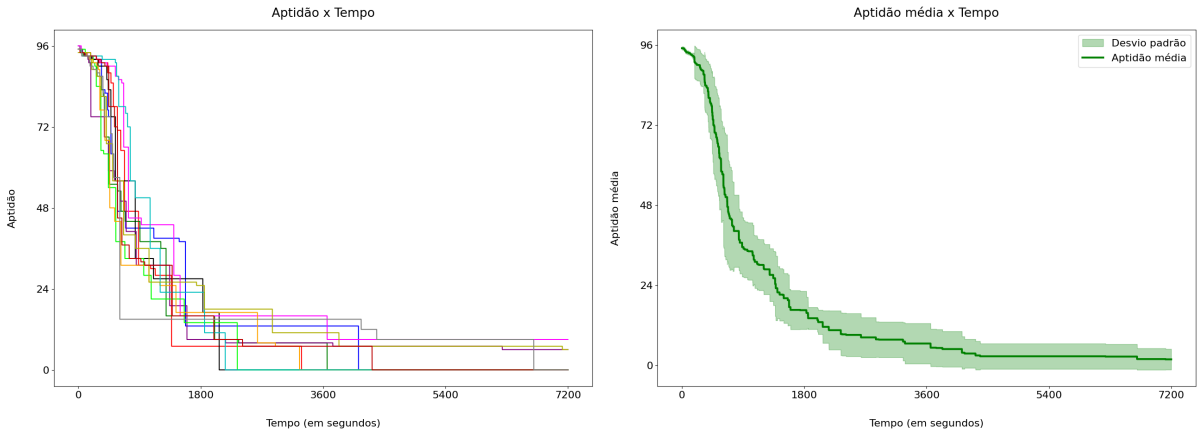
Fonte: O autor

Figura 4.12: Desempenho da mutação de Rechenberg no Simulador de Corrida ($D = 1$)



Fonte: O autor

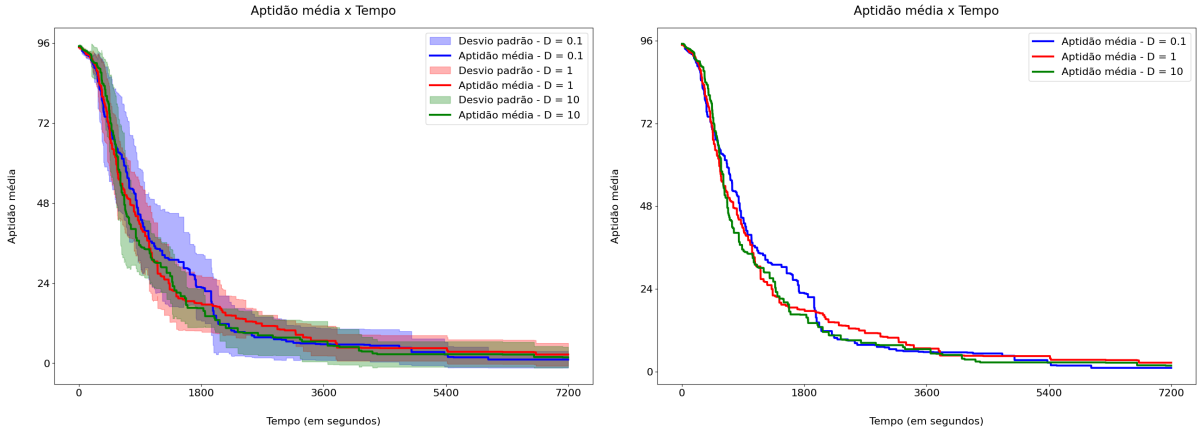
Figura 4.13: Desempenho da mutação de Rechenberg no Simulador de Corrida ($D = 10$)



Fonte: O autor

Para identificar qual valor apresentou o melhor desempenho, os gráficos exibidos nas figuras 4.11, 4.12 e 4.13 foram sobrepostos e comparados. A figura 4.14 exibe esta sobreposição. O gráfico à direita é equivalente ao gráfico à esquerda, porém, sem o sombreado que representa o desvio padrão dos resultados.

Figura 4.14: Comparação do desempenho da Mutação de Rechenberg (com $D = 0.1$, 1 e 10)



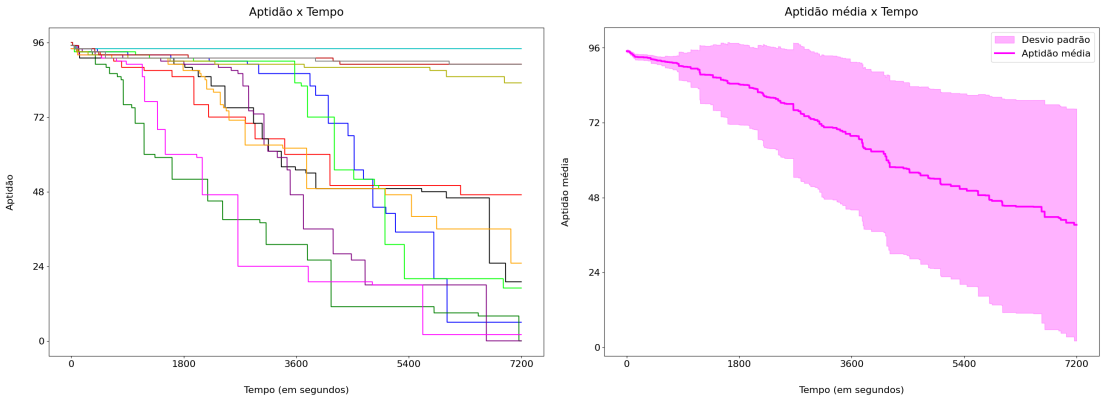
Fonte: O autor

Analisando o gráfico exibido na figura 4.14, pode-se notar que os três valores apresentaram desempenhos muito semelhantes. A variação do valor do parâmetro D não influenciou significativamente a qualidade dos indivíduos encontrados nem a velocidade de convergência do processo evolutivo. Dessa forma, por apresentar um resultado ligeiramente melhor, o valor $D = 0.1$ foi escolhido para ser comparado ao valor m que apresentar o melhor desempenho.

Experimentando a mutação Parcial

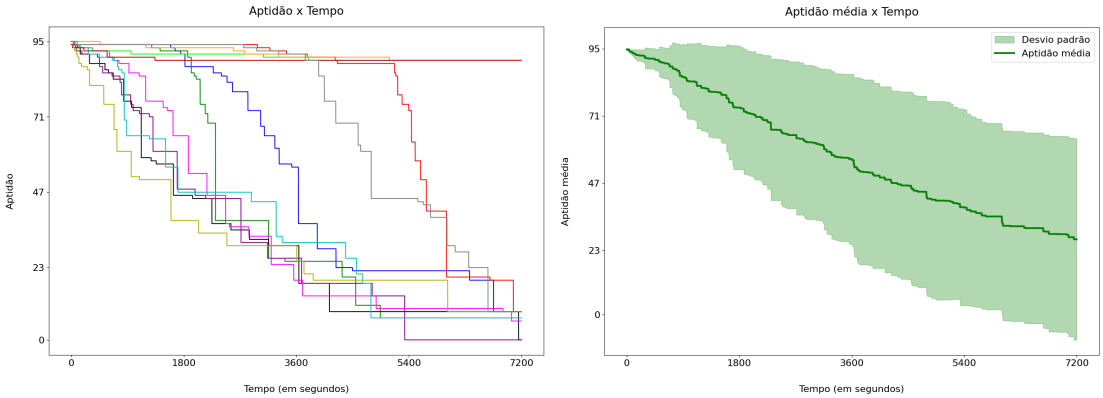
As figuras 4.15, 4.16, 4.17, 4.18, 4.19 e 4.20 exibem os resultados da aplicação da mutação parcial no Simulador de Corrida. Nestes experimentos, o valor do parâmetro m assumiu os seguintes valores: 2, 4, 8, 16, 32 e 64.

Figura 4.15: Desempenho da mutação Parcial no Simulador de Corrida ($m = 2$)



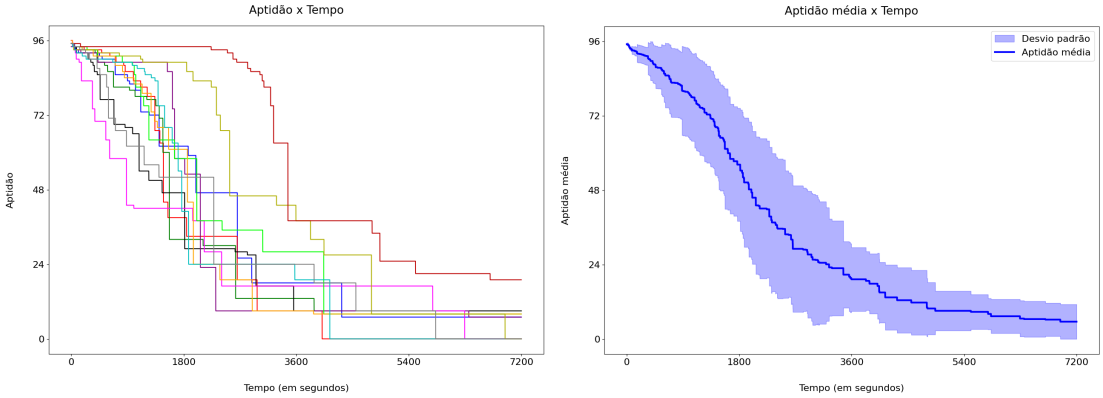
Fonte: O autor

Figura 4.16: Desempenho da mutação Parcial no Simulador de Corrida ($m = 4$)



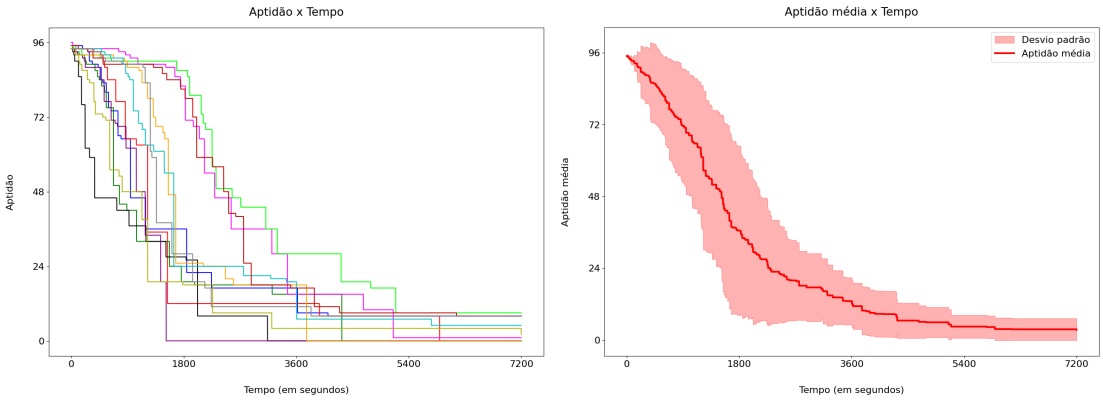
Fonte: O autor

Figura 4.17: Desempenho da mutação Parcial no Simulador de Corrida ($m = 8$)



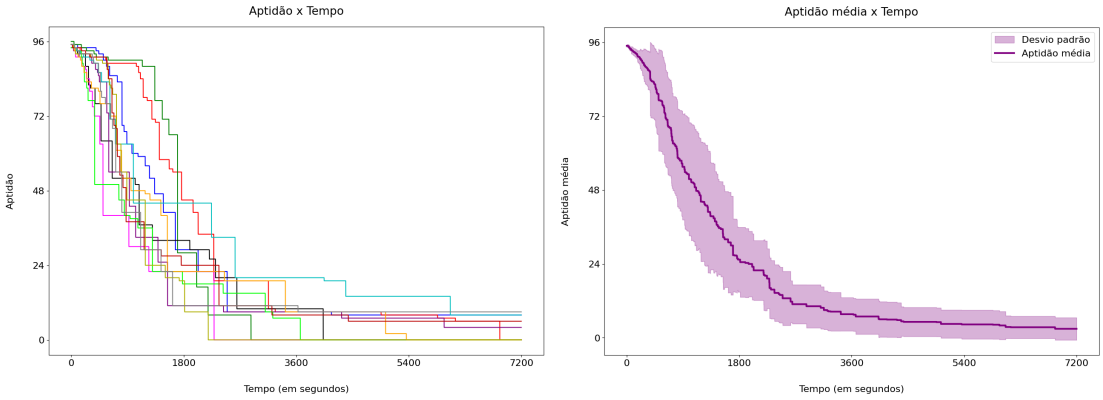
Fonte: O autor

Figura 4.18: Desempenho da mutação Parcial no Simulador de Corrida ($m = 16$)



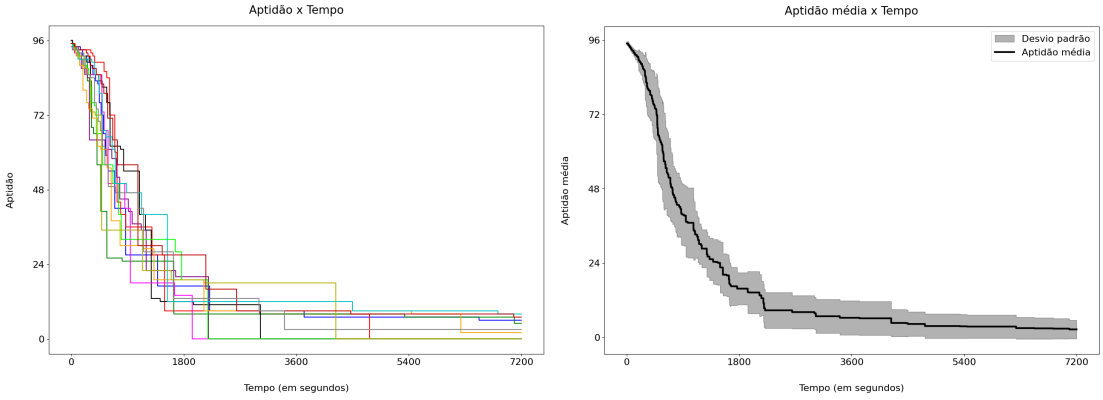
Fonte: O autor

Figura 4.19: Desempenho da mutação Parcial no Simulador de Corrida ($m = 32$)



Fonte: O autor

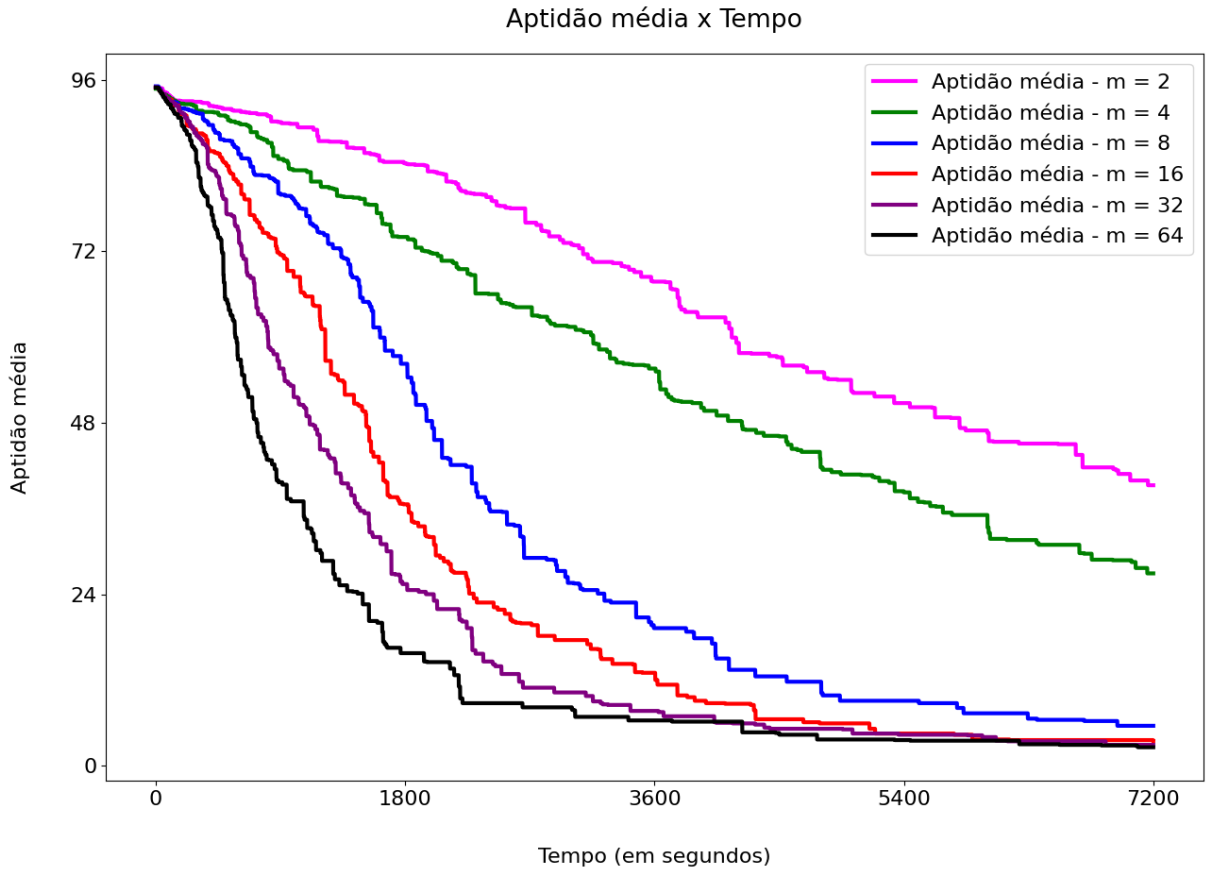
Figura 4.20: Desempenho da mutação Parcial no Simulador de Corrida ($m = 64$)



Fonte: O autor

Para identificar qual valor apresentou o melhor resultado, os gráficos exibidos nas figuras 4.15, 4.16, 4.17, 4.18, 4.19 e 4.20 foram sobrepostos e comparados. A figura 4.21 exibe esta sobreposição. Visando auxiliar a visualização, o sombreado que representa o desvio padrão foi retirado inserido.

Figura 4.21: Comparação do desempenho da Mutação Parcial (m = 2, 4, 8, 16, 32 e 64)



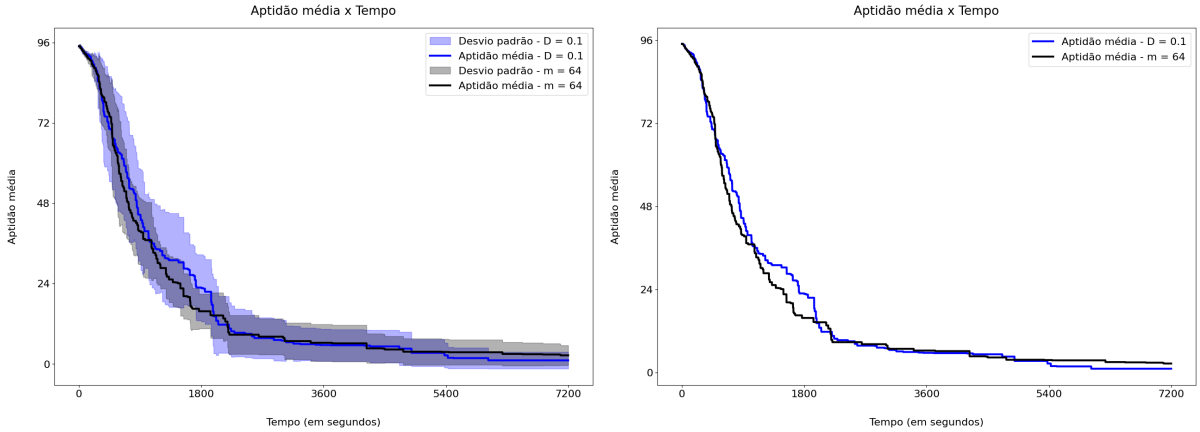
Fonte: O autor

Analisando o gráfico da figura 4.21, pode-se notar que, tanto a qualidade dos indivíduos encontrados, quanto a velocidade de convergência do método, aumentaram à medida que m aumentou. Dessa forma, por apresentar o melhor resultado, o valor $m = 64$ foi escolhido para ser comparado ao valor $D = 0.1$.

Comparação entre a mutação de Rechenberg e a mutação Parcial

Para identificar qual mutação apresentou o melhor resultado no Simulador de Corrida, o gráfico da mutação de Rechenberg onde $D = 0.1$ foi sobreposto ao gráfico da mutação parcial onde $m = 64$. A figura 4.22 exibe esta sobreposição.

Figura 4.22: Comparação entre o desempenho da mutação de Rechenberg e da mutação parcial



Fonte: O autor

Analisando os gráficos da figura 4.22, pode-se perceber que o desempenho dos métodos foi substancialmente semelhante. Porém, ao final do treinamento, a mutação de Rechenberg apresentou um resultado ligeiramente melhor. Além disso, o parâmetro D apresentou pouco (ou nenhum) impacto no resultado final do processo evolutivo, tornando-o assim, um parâmetro mais prático de se utilizar. Por esses motivos, a mutação de Rechenberg (com $D = 0.1$) foi escolhida para ser o método de mutação utilizado nos próximos experimentos.

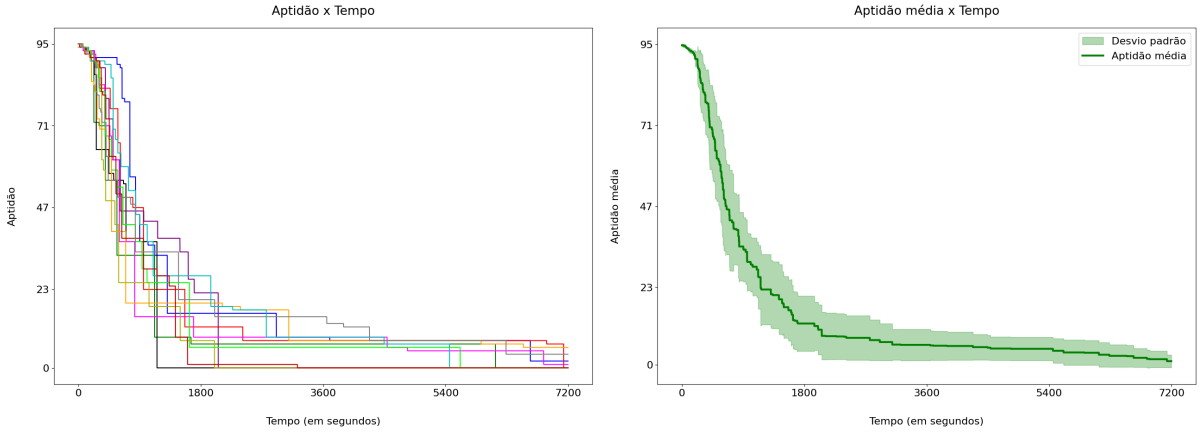
4.1.3 Experimentando Auto-Adaptação

Nesta subseção, serão apresentados os resultados da avaliação do impacto do mecanismo de Auto-Adaptação no desempenho do processo evolutivo. O mecanismo de Auto-Adaptação consiste em incluir o valor do parâmetro D ao processo de otimização. Dessa forma, este valor também é ajustado ao longo do processo evolutivo.

Este experimento foi realizado nas mesmas condições e com os mesmos parâmetros que o experimento utilizado para avaliar o desempenho da mutação de Rechenberg com D = 0.1. Porém, desta vez, o mecanismo de Auto-Adaptação esteve ativado durante o

processo. A figura 4.23 exibe os resultados deste experimento .

Figura 4.23: Desempenho da mutação de Rechenberg ($D = 0.1$) com Auto-Adaptação ativada

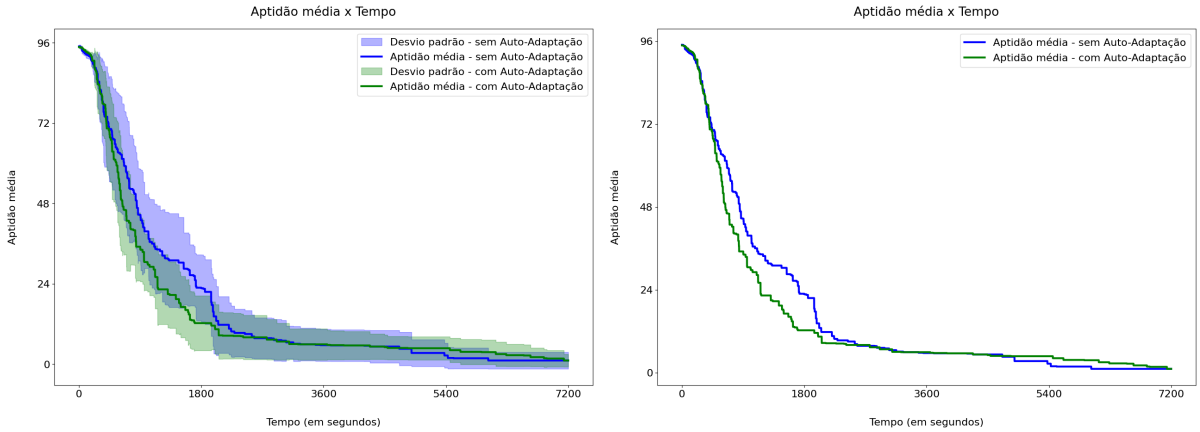


Fonte: O autor

Comparação dos experimentos com e sem a Auto-Adaptação

Para avaliar o impacto da ativação do mecanismo de Auto-Adaptação no processo evolutivo, o gráfico exibido na figura 4.23 foi sobreposto ao gráfico exibido na figura 4.11. A figura 4.24 exibe esta sobreposição.

Figura 4.24: Comparação do desempenho da Mutação de Rechenberg ($D = 0.1$) com e sem Auto-Adaptação



Fonte: O autor

Analisando o gráfico da figura 4.24, pode-se perceber que, apesar de apresentar uma velocidade de convergência ligeiramente maior no inicío do treinamento, o mecanismo de Auto-Adaptação não demonstrou grande impacto no resultado final do processo

evolutivo. Além disso, a adição deste mecanismo torna a implementação e a execução do processo mais complexa. Por esses motivos, foi preferido não ativar o mecanismo de Auto-Adaptação nos próximos experimentos.

4.2 Treinamento

Para obter redes neurais com desempenho elevado, foram realizados 4 experimentos (um para cada jogo) compostos por um único treinamento com 24 horas de duração. Por dispor de uma quantidade maior de tempo, a quantidade de partidas que os indivíduos jogaram antes de terem suas aptidões calculadas pôde ser aumentada. Este aumento tem o objetivo de potencializar a capacidade de generalização da rede. Os valores foram definidos da seguinte maneira:

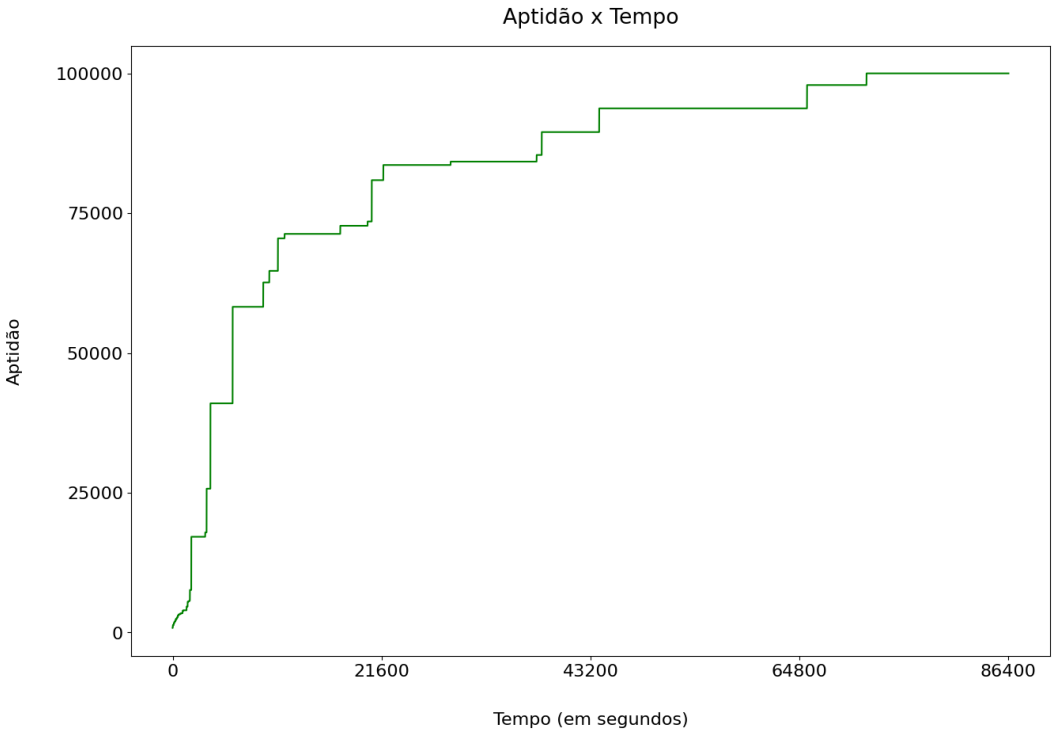
- *Flappy Bird* e Dinossauro do *Google Chrome*: 100 partidas
- Simulador de Foguetes: 1500 partidas (500 para cada tarefa)
- Simulador de Corridas: 50 partidas

Os parâmetros utilizados na estratégia evolutiva durante estes experimentos foram os parâmetros determinados na etapa de Exploração de Parâmetros:

- Método de seleção: (μ, λ)
- Método de mutação: mutação de Rechenberg (com D = 0.1)
- Auto-Adaptação: desligada.

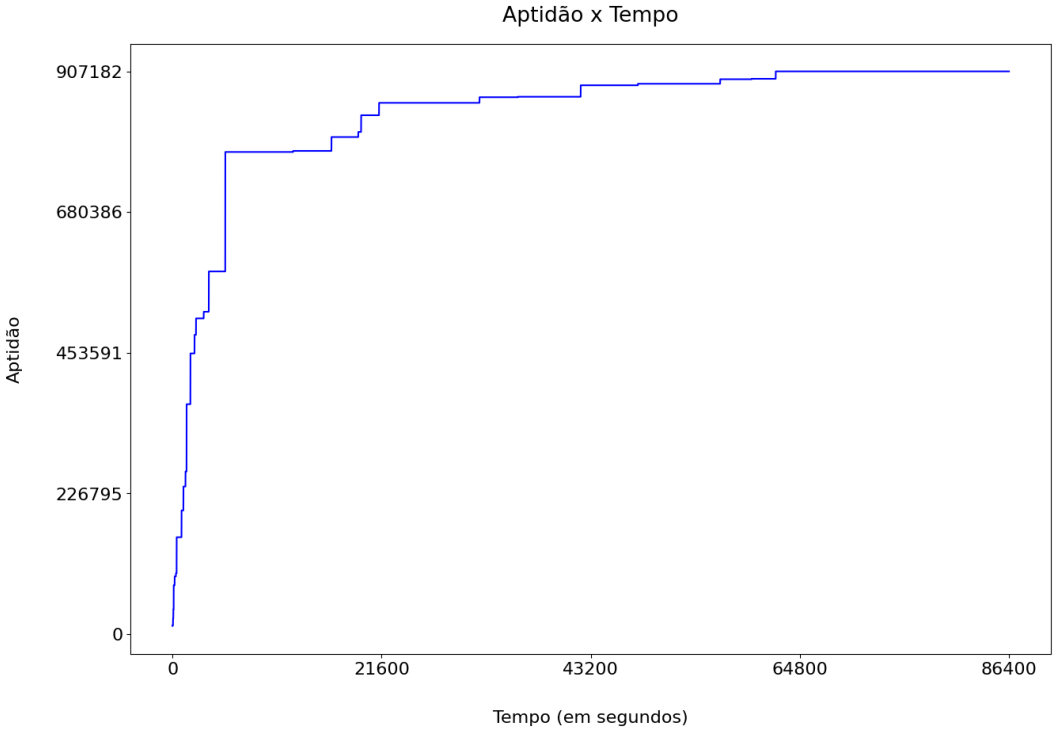
As figuras 4.25, 4.26, 4.27 e 4.28 exibem os resultados destes treinamentos em cada um dos jogos implementados.

Figura 4.25: Treinamento final: *Flappy Bird*



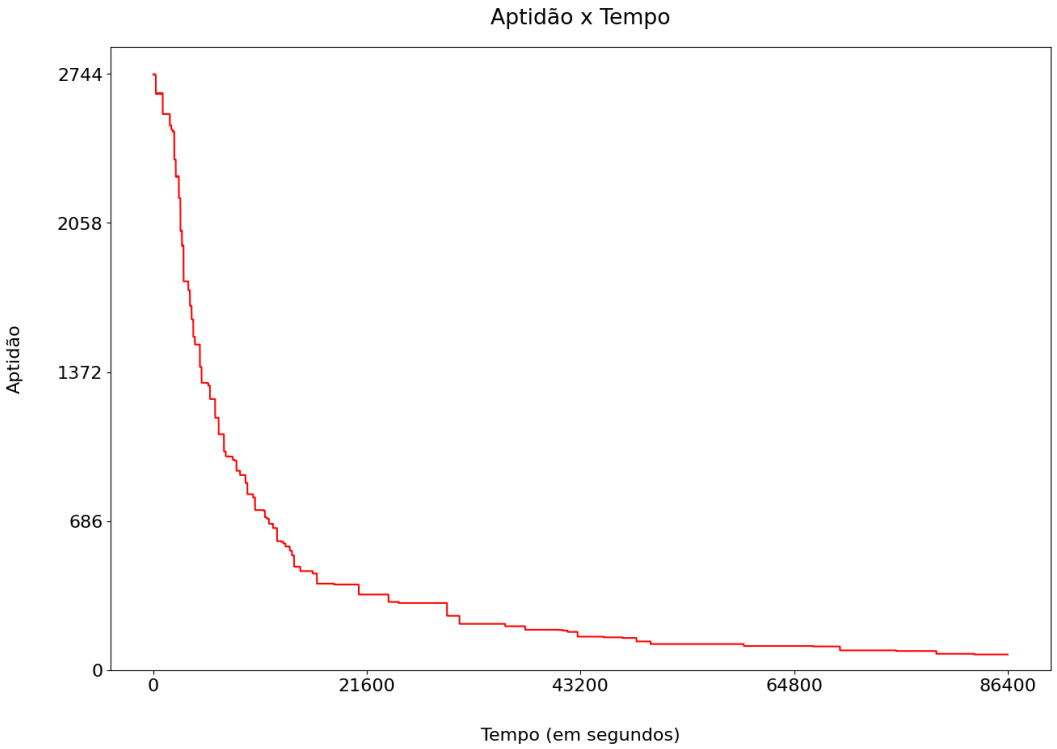
Fonte: O autor

Figura 4.26: Treinamento final: Dinossauro do *Google Chrome*



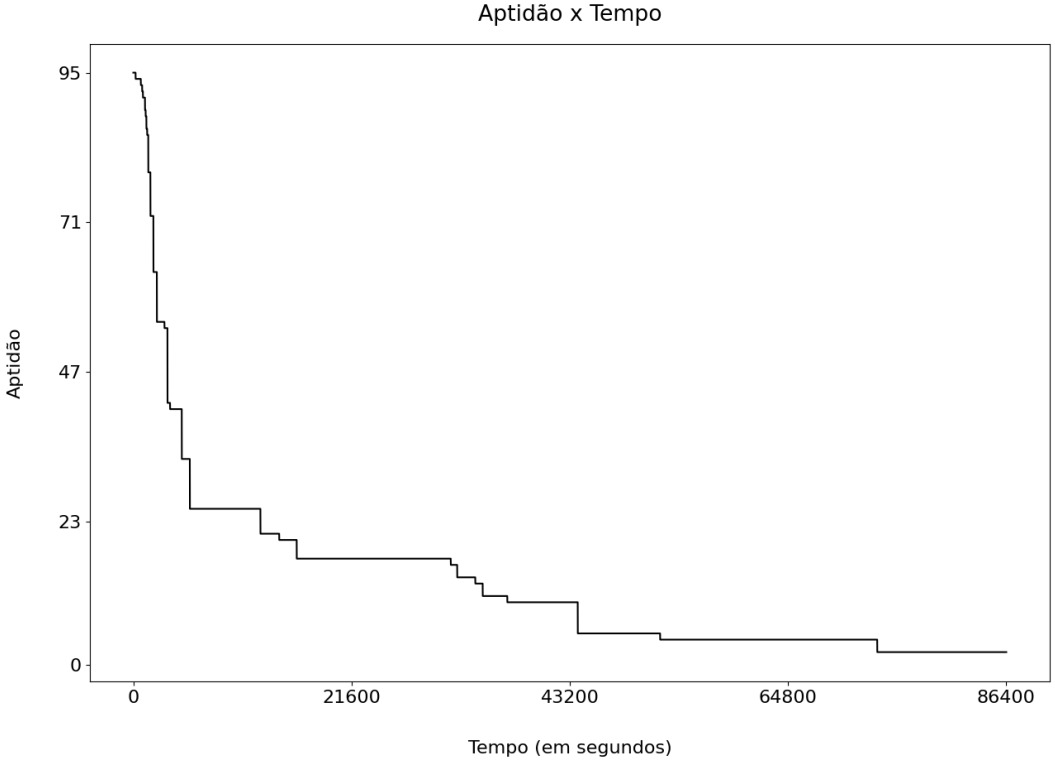
Fonte: O autor

Figura 4.27: Treinamento final: Simulador de Foguetes



Fonte: O autor

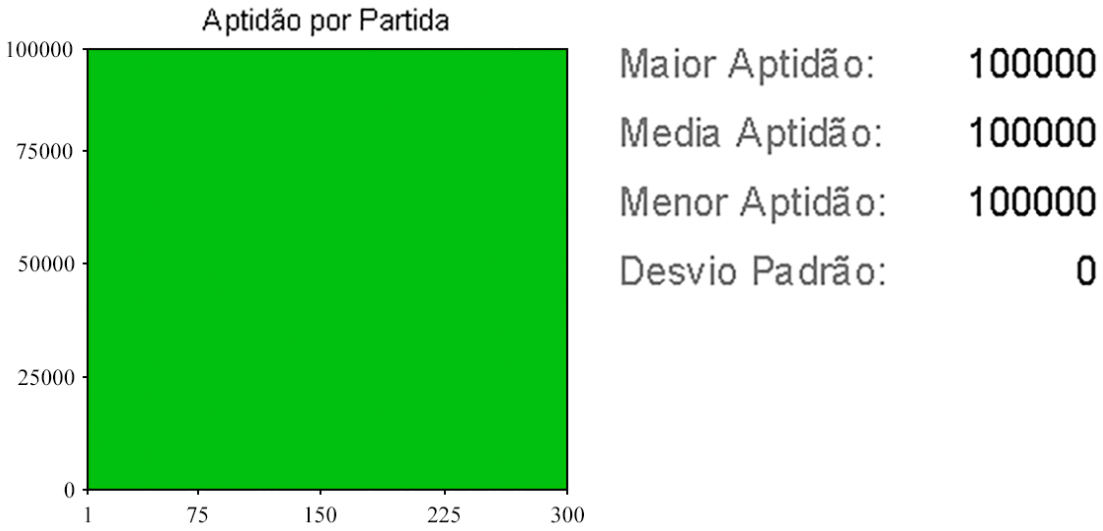
Figura 4.28: Treinamento final: Simulador de Corrida



Fonte: O autor

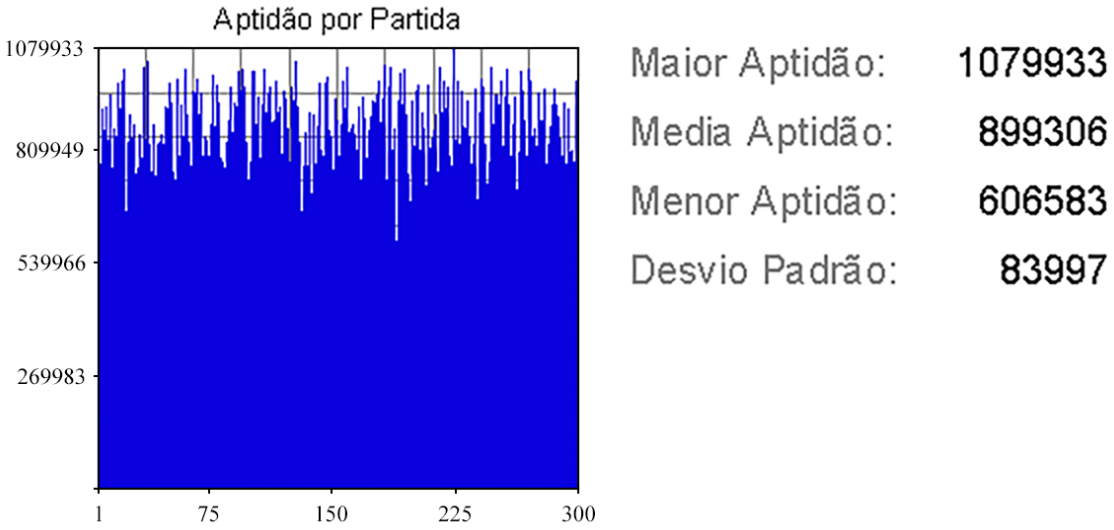
Por fim, as figuras 4.29, 4.30, 4.31 e 4.32 exibem gráficos de barras e informações sobre a etapa de validação da melhor rede neural obtida em cada jogo. Nesta etapa, cada rede foi colocada para jogar 300 partidas[1].

Figura 4.29: Validação final: Flappy Bird



Fonte: O autor

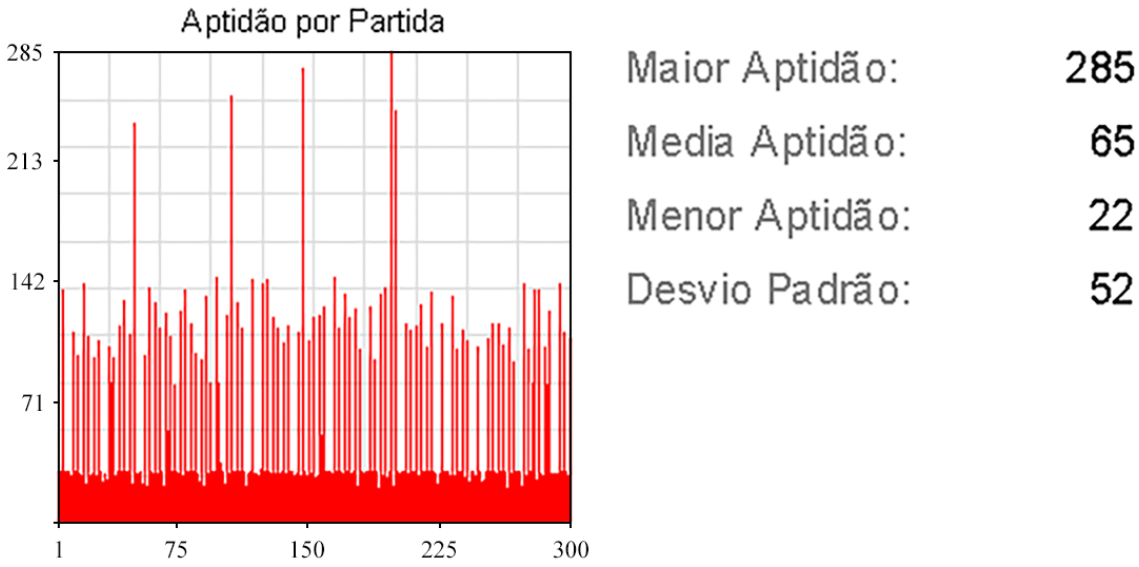
Figura 4.30: Validação final: Dinossauro do Google Chrome



Fonte: O autor

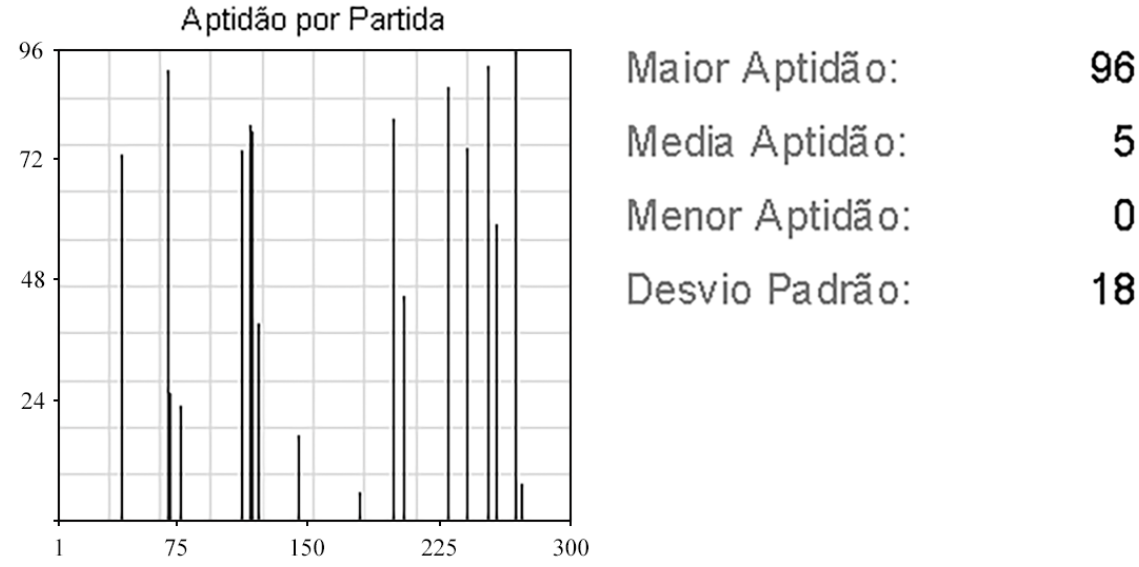
[1]Vídeos exibindo as redes neurais jogando cada um dos jogos estão disponíveis em: `https://youtube.com/channel/UCeltRClek1VcVVYOLNeGcoQ`

Figura 4.31: Validação final: Simulador de Foguetes



Fonte: O autor

Figura 4.32: Validação final: Simulador de Corrida



Fonte: O autor

Analisando os gráficos, pode-se notar que:

- De acordo a figura 4.29, no jogo *Flappy Bird*, a rede conseguiu atingir 100.000 pontos (*pixels* percorridos) em todas as 300 partidas jogadas por ela. Ou seja, o personagem

controlado pela rede neural conseguiu percorrer 30 milhões de *pixels* sem cometer nenhum erro, o que pode ser interpretado como uma acurácia de 100%.

- De acordo com a figura 4.30, no jogo do Dinossauro do *Google Chrome*, a rede atingiu uma média de 899.306 *pixels* percorridos nas 300 partidas jogadas por ela. Além disso, dentre todas as partidas, a menor aptidão alcançada foi 606.583. Isto significa que, no pior caso, a rede conseguiu sobreviver no jogo por 606.583 *pixels*. Este valor é equivalente a 331 segundos sobrevividos e aproximadamente 926 obstáculos ultrapassados.

- Na figura 4.31, as 5 barras que se destacam (as mais altas) representam as partidas nas quais a rede falhou ao pousar. Isto significa que, dentre 100 partidas destinadas ao pouso, a rede realizou o pouso com sucesso em 95 partidas, representando uma taxa de acerto de 95%.

- Na figura 4.32, as 18 barras visíveis representam as partidas nas quais a rede não conseguiu atingir a linha de chegada da pista. Isto significa que, dentre as 300 pistas jogadas, a rede finalizou com sucesso 282 pistas, representando uma taxa de acerto de 94%.

Capítulo 5

Conclusão

Este trabalho iniciou-se da seguinte pergunta de pesquisa: as estratégias evolutivas são eficazes quando utilizadas como método de treinamento para redes neurais perceptron no contexto de jogos digitais? Visando responder esta pergunta, estabeleceu-se a seguinte hipótese: sim, redes neurais perceptron conseguem atingir um elevado desempenho nos jogos digitais quando treinadas com estratégias evolutivas, visto que, as estratégias evolutivas são algoritmos de otimização estilo caixa-preta e por isso são ideais para o ambiente complexo e não-supervisionado dos jogos digitais.

Para testar a hipótese, foram implementadas as redes neurais, as estratégias evolutivas e os jogos selecionados. Após as implementações, estes algoritmos foram configurados e os treinamentos das redes neurais foram realizados. Após o término dos treinamentos, os resultados foram analisados e percebeu-se que a hipótese foi confirmada, tendo em vista que, de acordo com os resultados obtidos, as redes neurais conseguiram atingir uma acurácia em torno de 94% nos jogos implementados.

Por fim, para que o trabalho fosse realizado em tempo hábil, os experimentos envolvendo os métodos de mutação tiveram de ser realizados em apenas um jogo, o Simulador de Corrida. Além disso, foram implementados 4 jogos. Esta quantidade pode ser considerada uma amostra pequena para se determinar a eficácia do método em um contexto tão diverso quanto o dos jogos digitais. Diante destas limitações, algumas recomendações para trabalhos futuros podem ser estabelecidas:

- Realizar os 9 experimentos que envolvem os dois métodos de mutação em todos os 4 jogos.

- Adicionar mais jogos ao conjunto de jogos testados.
- Avaliar o impacto de outros parâmetros da estratégia evolutiva no resultado final do treinamento. Por exemplo: a quantidade de progenitores e a quantidade de descendentes.

Referências

[1] JOGO. *In*: Oxford Languages and Google - Portuguese. Oxford University Press, 2023.

[2] HUIZINGA, J. **Homo Ludens - O jogo como elemento da cultura**. Perspectiva S.A, 2000.

[3] SEBBANE, M. **Board Games from Canaan in the Early and Intermediate Bronze Ages and the Origin of the Egyptian Senet Game**. Tel Aviv, 28(2), 213-230, Routledge, 2001.

[4] PICCIONE, P. A. **The egyptian game of senet and the migration of the soul**. Ancient Board Games in Perspective, 54–63, British Museum Press, 2007.

[5] CRIST, W. **Passing from the Middle to the New Kingdom: A Senet Board in the Rosicrucian Museum**. The Journal of Egyptian Archaeology, 105(1), 107–113, 2019.

[6] Global Video Game Consumer Population Passes 3 Billion. **DFC Intelligence**, 2020. Disponível em <https://www.dfcint.com/global-video-game-consumer-population>. Acesso em: 04/06/2023.

[7] Que indústria fatura mais: do cinema, da música ou dos games?. **Super Interessante**, 2020. Disponível em <https://super.abril.com.br/mundo-estranho/que-industria-fatura-mais-do-cinema-da-musica-ou-dos-games>. Acesso em: 04/06/2023.

[8] Global Games Market to Generate $175.8 Billion in 2021; Despite a Slight Decline, the Market Is on Track to Surpass $200 Billion in 2023. **Newzoo**, 2021. Disponível em <https://newzoo.com/resources/blog/global-games-market-to-generate-175-8-billion-in-2021-despite-a-slight-decline-the-market-is-on-track-to-surpass-200-billion-in-2023>. Acesso em: 04/06/2023.

[9] BAKER, B. et al. **Emergent tool use from multi-agent autocurricula**. arXiv:1909.07528. 2019. Disponível em: <https://arxiv.org/abs/1909.07528>

[10] WON, J. et al. **Control Strategies for Physically Simulated Characters Performing Two-player Competitive Sports**. ACM Trans. Graph. 40(4), 1–11, 2021.

[11] BANSAL, T. **Emergent complexity via multi-agent competition**. arXiv:1710.03748. 2018. Disponível em: <https://arxiv.org/abs/1710.03748>

[12] MNIH, V. et al. **Human-level control through deep reinforcement learning**. Nature, 518, 529–533, 2015.

[13] CUNHA, A. G. **Manual de computação evolutiva e metaheurística**. Editora da Universidade Federal de Minas Gerais, 2013.

[14] WRIGHT, S. J. **Primal-Dual Interior-Point Methods**. Society for Industrial and Applied Mathematics, 1997.

[15] HILLIER, F.S.; LIEBERMAN, G.J. **Introdução à Pesquisa Operacional**. AMGH, 2013.

[16] NOCEDAL, J. WRIGHT, S. J. **Numerical Optimization**. Springer New York, 2006.

[17] HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. The MIT Press, 1992.

[18] GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. Addison-Wesley, 1989.

[19] Box, G. E. P.; Muller, M. E. **A note on the generation of random normal deviates**. Annals of Mathematical Statistics, 29(2), 610-611, 1958.

[20] RECHENBERG, I. **Evolutionsstrategie - Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution**. Frommann-Holzboog, 1973.

[21] SCHWEFEL, H. P. **Numerical Optimization of Computer Models**. Wiley, 1981.

[22] FACELI, K. et al. **Inteligência Artificial: Uma abordagem de Aprendizado de Máquina**. Livros Técnicos e Científicos Editora LTDA, 2011.

[23] ROSENBLATT, F. **The perceptron: a probabilistic model for information storage and organization in the brain**. Psychological Reviews, 65(6), 386–408, 1958.

[24] MCCULLOCH, W. S.; PITTS, W. **A logical calculus of the ideas in nervous activity**. Bulletin of Mathematical Biophysics, 5, 115-133, 1943.

[25] CYBENKO, G. **Approximation by superpositions of a sigmoid function**. Math. Control Signal Systems, 2, 303–314, 1989.

[26] RUMELHART, D. E. et al. **Learning internal representations by error propagation**. Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations, MIT Press, 1986.

[27] HEBB, D. O. **The Organization of Behavior**. Psychology Press, 2002.

[28] WATKINS, C. **Learning from Delayed Rewards**. University of Cambridge, 1989.

[29] MNIH, V. **Playing Atari with Deep Reinforcement Learning**. arXiv:1312.5602, 2013. Disponível em: <https://arxiv.org/abs/1312.5602>

[30] GLOROT, X.; YOSHUA, B. **Understanding the difficulty of training deep feedforward neural networks**. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 9, 249-256, 2010.

[31] HE, K. **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**. arXiv:1502.01852, 2015. Disponível em: <https://arxiv.org/abs/1502.01852>

[32] BRAGA, A. P. et al. **Redes Neurais Artificiais: teoria e aplicações**. LTC, 2007.

[33] HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. Prentice-Hall, 1999.

[34] ROSS, S. **Probabilidade - Um Curso Moderno com Aplicações**. Bookman, 2010.

[35] BERNOULLI, J. **Ars Conjectandi**. Thurnisiorum, 1713.

[36] CHAGAS, C. S. et al. **Utilização de redes neurais artificiais na classificação de níveis de degradação em pastagens**. Revista Brasileira de Engenharia Agrícola e Ambiental, 13(3), 319–327, 2009.

[37] OLESKOVICZ, M. **O emprego de redes neurais artificiais na detecção, classificação e localização de faltas em linhas de transmissão**. Revista Controle e Automação, 14(2), 2003.

[38] COSTA, J. P. de O.; ORTOLAN, L. D. **Classificação de imagens de sensoriamento remoto utilizando redes neurais artificiais**. Universidade Tecnológica Federal do Paraná, 2014.

[39] RIBEIRO, S. R. A.; CENTENO, J. S. **Classificação do Uso do Solo Utilizando Redes Neurais e o Algoritmo MAXVER**. Anais X SBSR, 1341-1348, 2001.

[40] LECUN, Y. et al. **Gradient-based learning applied to document recognition**. Proceedings of the IEEE, 86(11), 2278-2324, 1998.

[41] SHAKER, N. et al. **Procedural Content Generation in Games**. Springer, 2009.

[42] HELSGAUN, K. **General k-opt submoves for the Lin–Kernighan TSP heuristic**. Math. Prog. Comp. 1, 119–163, 2009.

[43] CATMULL, E.; ROM, R. **A class of local interpolating splines**. Academic Press, Computer Aided Geometric Design, 317-326, 1974.

[44] LI, Y.; IBANEZ-GUZMAN, J. **Lidar for Autonomous Driving: The principles, challenges, and trends for automotive lidar and perception systems**. IEEE Signal Processing Magazine, 37(4), 50-61, 2020.

[45] CHOSET, H. et al. **Principles of robot motion: theory, algorithms, and implementations**. MIT Press, 2005.