UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Peres Mundim

Aplicação de heurísticas computacionais bioinspiradas no jogo do dinossauro

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Guilherme Peres Mundim

Aplicação de heurísticas computacionais bioinspiradas no jogo do dinossauro

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Profa. Dra. Christiane Regina Soares Brasil

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Guilherme Peres Mundim

Aplicação de heurísticas computacionais bioinspiradas no jogo do dinossauro

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 24 de setembro de 2025:

Profa. Dra. Christiane Regina Soares Brasil
Orientadora

Profa. Dra. Fernanda Maria da Cunha Santos
Professora

Profa. Dra. Daniela Justiniano de Sousa
Professora

Uberlândia, Brasil
2025

Resumo

A Computação Bioinspirada refere-se a uma área de pesquisa dedicada ao desenvolvimento de heurísticas computacionais inspiradas em fenômenos naturais, com o objetivo de lidar com problemas altamente complexos. Neste Trabalho de Conclusão de Curso, duas heurísticas computacionais foram implementadas – o Algoritmo Evolutivo (AE) e a Rede Neural Artificial – para a solução do jogo do dinossauro do Google Chrome, baseado na pesquisa de Pereira (2023). Neste jogo, os dinossauros devem escapar dos obstáculos, tentando sobreviver o máximo de tempo possível. Neste contexto, as técnicas inteligentes foram aplicadas para que os dinossauros pudessem agir de modo autônomo. Os resultados encontrados nesta pesquisa foram promissores, e quando comparados à literatura, apresentaram valores superiores nas métricas de avaliação usadas, no caso, os valores de aptidão de cada solução.

Palavras-chave: Inteligência Artificial, Redes Neurais Artificiais, Algoritmos Evolutivos, Jogos Digitais, Dinossauro do Google.

Abstract

Bioinspired Computing refers to a research area dedicated to the development of computational heuristics inspired by natural phenomena, aiming to address highly complex problems. In this Final Course Work, two computational heuristics were implemented—the Evolutionary Algorithm (EA) and the Artificial Neural Network—to solve the dinosaur game in Google Chrome, based on the research of Pereira (2023). In this game, dinosaurs must escape obstacles, trying to survive as long as possible. In this context, intelligent techniques were applied so that the dinosaurs could act autonomously. The results found in this work were promising, and when compared to the literature, they presented superior values in the evaluation metrics used, in this case, the fitness values of each solution.

Keywords: Artificial Intelligence, Artificial Neural Networks, Evolutionary Algorithms, Digital Games, Google Dinosaur.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IA	Inteligência Artificial
RNA	Rede Neural Artificial
MLP	Perceptron Multicamadas
AG	Algoritmo Genético
AE	Algoritmo Evolutivo
EE	Estratégias Evolutivas
ReLU	Unidade Linear Retificada
TCC	Trabalho de Conclusão de Curso
JDK	Java Development Kit
IDE	Ambiente Integrado de Desenvolvimento
SO	Sistema Operacional
CPU	Unidade Central de Processamento
RAM	Memória de Acesso Aleatório
JVM	Máquina Virtual Java
FPS	Quadros por segundo

Sumário

1 Introdução

A Inteligência Artificial (IA) tem desempenhado um papel revolucionário nos jogos contemporâneos, automatizando processos e reinventando a forma como as experiências são vivenciadas. É notável como a IA contribui, de fato, para tornar os jogos mais dinâmicos e imersivos para os usuários. Além disto, os jogos são um ambiente propício à experimentação e ao aperfeiçoamento de tais técnicas computacionais aplicadas, justamente por serem classificados, em geral, como problemas de alta complexidade.

Neste cenário, os Algoritmos Evolutivos (AEs) e as Redes Neurais Artificiais (RNAs) têm se destacado como abordagens promissoras para o desenvolvimento de agentes inteligentes em jogos. Inspirados na teoria da evolução de Darwin (DARWIN, 1859), os AEs (GABRIEL; DELBEM, 2008) utilizam processos que mimetizam a reprodução de indivíduos – como a recombinação, a clonagem e a mutação – com o objetivo de encontrar a solução ótima para um dado problema ou aproximações para a mesma (LIMA; PINHEIRO; SANTOS, 2014). Em jogos, isto possibilita a criação de personagens capazes de aprender e se adaptar ao ambiente de forma autônoma, aumentando tanto o desafio para o jogador quanto o valor científico do jogo como laboratório para ambientes complexos.

Por outro lado, as RNAs simulam o funcionamento do cérebro humano e demonstram grande capacidade para processar dados e identificar padrões (LIMA; PINHEIRO; SANTOS, 2014). Em jogos, isto permite a criação de agentes que não apenas reagem a estímulos do ambiente, mas também aprendem com experiências anteriores, evoluindo ao longo do tempo (RAHMAN, 2024). Quando combinadas com os AEs, estas redes ampliam as possibilidades de adaptação progressiva dos agentes, tornando a experiência do jogador mais imprevisível e envolvente.

Este Trabalho de Conclusão de Curso propôs o desenvolvimento de um jogo inspirado na pesquisa de Pereira (2023), que implementou o jogo do dinossauro do *Google Chrome*, um clássico de corrida infinita em que o personagem precisa desviar de obstáculos. Pereira (2023) recriou o jogo utilizando a linguagem C e a biblioteca PIG, incorporando uma Rede Neural Artificial (RNA) para controlar o dinossauro de forma autônoma. Em seu trabalho, foi utilizada um Perceptron *Multilayer* de três camadas, treinada por meio de um AE com mutações nos pesos da rede neural.

Portanto, este Trabalho de Conclusão de Curso foi baseado no estudo de Pereira (2023), ampliando-o com a aplicação de conceitos de orientação a objetos em Java, a experimentação de diferentes arquiteturas de Redes Neurais Artificiais, que controlam os movimentos de cada agente (dinossauro), em conjunto com a exploração de parâmetros

dos Algoritmos Evolutivos, responsáveis pelo ajuste dos pesos das RNAs.

1.1 Objetivo

O objetivo principal deste Trabalho de Conclusão de Curso foi desenvolver um sistema para o jogo do dinossauro do *Google Chrome* utilizando Algoritmos Evolutivos e Redes Neurais Artificiais. Baseado no estudo de Pereira (2023), os dinossauros deste jogo foram capazes de atuar de forma autônoma no ambiente, por meio das Redes Neurais Artificiais. O Algoritmo Evolutivo foi utilizado para ajustar os pesos/vieses das RNAs de cada dinossauro durante o processo evolutivo.

Os objetivos específicos deste trabalho foram:

- O desenvolvimento da interface gráfica utilizando Java;
- A criação e a estruturação do jogo, incluindo a mecânica do dinossauro, dos obstáculos e da lógica de pontuação;
- A exploração de diferentes arquiteturas de Redes Neurais Artificiais, variando a quantidade de camadas;
- A implementação de um Algoritmo Evolutivo e a sua análise a partir dos valores de aptidão dos dinossauros obtidos em experimentos;
- A análise do impacto do ajuste de parâmetros do Algoritmo Evolutivo, como a taxa de mutação e o tamanho da população.
- A análise comparativa do desempenho de dois Algoritmos Evolutivos – o AE desenvolvido neste trabalho e o AE de um trabalho de referência – cujos processos evolutivos foram implementados de modo diferente.

1.2 Justificativa

A aplicação de técnicas bioinspiradas de Inteligência Artificial em jogos justifica-se pelo seu potencial em desenvolver personagens mais desafiadores, adaptáveis e responsivos, por exemplo, contribuindo de forma significativa para a melhoria da experiência de jogabilidade. Essa abordagem estimula a pesquisa em IA aplicada a jogos, fomentando inovações na criação de agentes inteligentes capazes de atuar em ambientes complexos e dinâmicos.

Esses agentes, ao utilizarem heurísticas computacionais em vez de estratégias determinísticas, demonstram maior capacidade de lidar com incertezas e variabilidades do ambiente. Isso não apenas eleva o nível de realismo e imersão nos jogos, mas também

impulsiona o desenvolvimento de métodos computacionais avançados, com aplicações potenciais em áreas como robótica, simulações autônomas e sistemas de apoio à decisão.

1.3 Organização do texto

Este trabalho está organizado como descrito a seguir:

- **Capítulo 1 – Introdução**: apresenta o problema, os objetivos do estudo e a justificativa, além deste roteiro de organização do texto.
- **Capítulo 2 – Fundamentação Teórica**: reúne os conceitos utilizados ao longo do desenvolvimento deste Trabalho de Conclusão de Curso. São abordados assuntos como: otimização computacional, heurísticas, conceitos sobre Algoritmo Evolutivo e os fundamentos de Rede Neural Artificial.
- **Capítulo 3 – Trabalhos Correlatos**: sintetiza os principais estudos de referência e apresenta uma comparação que posiciona este trabalho frente à literatura.
- **Capítulo 4 – Desenvolvimento**: detalha a construção do sistema. Inclui a referência principal utilizada para a implementação do jogo, a organização da implementação, a modelagem do dinossauro e dos inimigos, a implementação da Rede Neural Artificial, do Algoritmo Evolutivo e da interação entre RNA, dinossauro e cenário, além das principais diferenças em relação à implementação de Pereira (2023).
- **Capítulo 5 – Resultados**: discute os resultados obtidos a partir dos experimentos realizados para a definição da arquitetura da RNA mais adequada e dos melhores parâmetros para o AE aplicados ao problema em questão. Além disto, é exposta uma análise comparativa do AE desenvolvido nesta pesquisa com uma investigação de referência.
- **Capítulo 6 – Conclusão**: apresenta as conclusões principais a partir dos resultados e as sugestões de desdobramentos futuros.

2 Fundamentação Teórica

Este capítulo revisará os principais conceitos utilizados para a elaboração deste Trabalho de Conclusão de Curso, incluindo noções de Otimização Computacional e heurísticas e, de forma mais aprofundada, os Algoritmos Evolutivos e as Redes Neurais Artificiais, bem como apresentará alguns trabalhos relacionados ao tema.

A seguir, será descrita a fundamentação teórica sobre a área de Otimização Computacional e heurísticas.

2.1 Otimização Computacional

A otimização computacional é uma área da pesquisa operacional que se dedica a encontrar a melhor solução possível para um problema em que seria inviável buscá-la por métodos determinísticos (LACHTERMACHER, 2001). Em termos práticos, ela busca utilizar da melhor forma os recursos disponíveis, seja para reduzir custos, aumentar lucros ou melhorar a eficiência de um processo.

Taha (2008) descreve a otimização como a busca do valor máximo ou mínimo de uma função matemática chamada função-objetivo. Esta função depende de variáveis de decisão, que representam elementos ajustáveis pelo decisor, e de restrições, que definem os limites das soluções possíveis. Deste modo, a solução ótima é aquela que oferece o melhor resultado dentro de todo o espaço de alternativas viáveis.

Historicamente, os primeiros estudos de otimização remontam à década de 1930, associados ao desenvolvimento do radar na Inglaterra. Desde então, a área cresceu e consolidou-se, com a criação de sociedades especializadas como a *Operations Research Society of America* (ORSA), em 1952, e a Sociedade Brasileira de Pesquisa Operacional (SOBRAPO), em 1969. Essas instituições reforçam a importância da otimização em aplicações práticas que vão desde teoria de filas e fluxo em redes até planejamento de estoques e reposição de equipamentos.

De forma geral, a resolução de um problema de otimização segue três etapas principais:

1. **Formulação do problema**: a definição do objetivo, das variáveis de decisão, das restrições e dos elementos fora do controle dos decisores;
2. **Construção do modelo**: a representação matemática que descreve as relações entre as variáveis e os dados relevantes;

3. **Obtenção da solução e implementação**: a aplicação de métodos matemáticos ou heurísticos para encontrar a melhor solução, seguida da análise e implementação no contexto prático.

Esses modelos matemáticos podem ser heurísticas computacionais, as quais serão explicadas na seguinte seção.

2.2 Heurística Computacional

A heurística computacional é uma técnica que busca a solução ótima ou sub-ótima em um tempo hábil para problemas de alta complexidade, onde métodos exatos seriam inviáveis devido ao grande custo computacional (PEARL, 1984). Tais problemas podem ser classificados computacionalmente como NP[1] e suas variantes (GAREY; JOHNSON, 1979). Alguns exemplos clássicos de problemas NP são o Problema do Caixeiro Viajante, o Problema da Mochila e a Coloração de Grafos.

Portanto, a principal característica de uma heurística é a busca por soluções ótimas ou satisfatórias de forma eficiente utilizando métodos não-determinísticos. Em muitos cenários, uma resposta razoável obtida em segundos é mais útil que a solução perfeita que exigiria horas ou dias de cálculo. Neste sentido, os Algoritmos Evolutivos (AEs) podem ser classificados como heurísticas computacionais. Inspirados na teoria da evolução de Darwin, eles tratam cada solução como um indivíduo que compete pela sobrevivência com base em sua aptidão. A busca pela solução ótima ou sub-ótima é modelada como um processo de seleção natural, em que os indivíduos mais aptos têm maior probabilidade de sobreviver e gerar descendentes que herdam suas características.

Os principais conceitos relacionados aos AEs serão aprofundados na próxima seção.

2.3 Algoritmo Evolutivo

Os Algoritmos Evolutivos (AEs) (GABRIEL; DELBEM, 2008) são técnicas de computação bioinspirada, ou seja, métodos que mimetizam os processos naturais de evolução, descritos por Charles Darwin em sua obra "A Origem das Espécies" (DARWIN, 1859). Assim como na natureza, em que os organismos mais adaptados possuem maior chance de sobreviver, os AEs utilizam esse mesmo princípio para resolver problemas complexos, buscando soluções cada vez melhores ao longo de várias gerações (GASPAR-CUNHA; ANTUNES; TAKAHASHI, 2012). Esta capacidade de adaptação torna os AEs adequados para cenários de alta complexidade computacional.

[1] Um problema NP (do inglês *Nondeterministic Polynomial time*) pode ser resolvido em tempo polinomial por uma máquina de Turing não determinística.

Por definição, cada indivíduo em um AE representa uma solução possível para o problema em questão. Estes indivíduos formam uma população inicial que, geração após geração, é submetida a processos inspirados na biologia, como seleção, cruzamento e mutação. Os indivíduos mais bem adaptados (ou seja, as melhores soluções) tendem a permanecer ou serem escolhidos para o cruzamento, propiciando a evolução da população. Com isso, o algoritmo consegue evoluir gradualmente em direção às soluções de maior qualidade.

Os AEs consolidaram-se a partir dos anos 1960 em três correntes principais da computação evolutiva: Programação Evolutiva, Algoritmos Genéticos e Estratégias Evolutivas (JONG, 2000). A Programação Evolutiva foi proposta por Fogel (1966), em San Diego; os Algoritmos Genéticos foram desenvolvidos por Holland (1967) e seus alunos em Ann Arbor; e as Estratégias Evolutivas tiveram origem em trabalhos em Berlim na década de 1960, notadamente com Rechenberg e Schwefel.

Estas três vertentes são brevemente descritas a seguir:

- Programação Evolutiva (PE): ênfase histórica em mutação e seleção, com foco original em modelos de comportamento (autômatos) e, posteriormente, em representações numéricas.
- Algoritmos Genéticos (AGs): uso obrigatório da seleção e da recombinação (*crossover*), sendo opcional a mutação sobre indivíduos, com forte papel da recombinação na exploração do espaço de busca.
- Estratégias Evolutivas (EEs): tradição em codificação real e variação controlada por parâmetros (p. ex., intensidades de mutação), muito usadas em otimização contínua.

Em todas as três abordagens, o conceito de população de indivíduos é fundamental para o funcionamento do algoritmo. A diversidade de indivíduos em uma população garante que ele explore diferentes regiões do espaço de busca, aumentando a chance de encontrar respostas ótimas ou próximas da ótima. Cada indivíduo pode ser representado por vetores numéricos ou estruturas equivalentes, que simbolizam sua configuração para o problema (GASPAR-CUNHA; ANTUNES; TAKAHASHI, 2012).

Ao longo das gerações, alguns indivíduos podem ser escolhidos como progenitores (indivíduos pais), que darão origem a descendentes. Esta escolha pode ser feita com base na qualidade da solução (aptidão) ou de forma aleatória, dependendo da estratégia empregada. Os descendentes herdam características de seus pais, podendo sofrer mutações que introduzem variações aleatórias, aumentando a diversidade genética da população e evitando que o algoritmo fique preso em ótimos locais (platôs).

Este ciclo é repetido até que um critério de parada seja atingido (o número máximo de gerações alcançado ou a obtenção de uma solução satisfatória), como mostrado no Algoritmo 1.

Algoritmo 1 Pseudocódigo de um Algoritmo Evolutivo Típico.

```
1: GERAR população inicial aleatoriamente
2: AVALIAR cada solução
3: repita
4:    SELECIONA indivíduos para reprodução
5:    REALIZA cruzamento entre os selecionados
6:    REALIZA mutação nos descendentes, caso necessário
7:    AVALIA fitness das novas soluções
8:    SELECIONA soluções para a próxima geração, se houver elistimo
9: até CONDIÇÃO DE PARADA satisfeita
```

A seguir, serão definidos os principais compontes de um Algoritmo Evolutivo.

2.3.1 *Fitness*

O conceito de *fitness* (ou aptidão) é central nos Algoritmos Evolutivos, funcionando como uma métrica que avalia o quão bem um indivíduo se adapta a um dado problema. Em outras palavras, o *fitness* mede a qualidade de uma solução em relação ao ambiente em que está inserida.

No contexto de problemas de otimização, o *fitness* está diretamente associado à função-objetivo. Em tarefas de maximização, soluções com maior *fitness* representam melhores desempenhos. Por outro lado, em tarefas de minimização, o mais apto é aquele que apresenta menor custo em relação ao critério definido (GABRIEL; DELBEM, 2008).

Rechenberg (1973) e Holland (1992) ressaltam que o indivíduo é avaliado pelo sucesso em se adaptar ao ambiente, reforçando que cada elemento de uma população é constantemente testado contra o ambiente e classificado de acordo com sua capacidade de sobreviver e se reproduzir.

2.3.2 Seleção

A seleção é o processo responsável por escolher os indivíduos (pais) que participarão da reprodução em Algoritmos Evolutivos. Estes indivíduos podem ser escolhidos de modo aleatório ou de acordo com valor de sua aptidão (*fitness*), priorizando os melhores. Neste segundo caso, a evolução da população é potencializada, uma vez que este método privilegia as soluções mais promissoras sem eliminar completamente a diversidade.

Depois que o *fitness* de cada indivíduo é avaliado, diferentes estratégias de seleção podem ser aplicadas. Goldberg (1989) introduziu o **método da roleta**, no qual cada

indivíduo pode ser selecionado de acordo com o seu valor de aptidão. Isso significa que, quanto melhor o *fitness*, maior a probabilidade de ser escolhido. Miller e Goldberg destacaram a seleção por torneio de N, onde um subconjunto de N indivíduos é escolhido aleatoriamente e o mais apto dentro desse grupo é selecionado para reprodução, e isto é repetido para escolher o outro indivíduo, que será o outro pai.

É importante observar que, em diferentes abordagens de AEs, o mecanismo de seleção pode variar. Nas Estratégias Evolutivas, por exemplo, existem os modelos (μ, λ)-EE e (μ +λ)-EE. No primeiro, apenas os descendentes competem pela sobrevivência, enquanto no segundo pais e filhos disputam as vagas da próxima geração. Essa distinção permite maior controle sobre a diversidade da população, como discutido nos trabalhos de Rechenberg (1973) e Schwefel (1975).

Além disso, existem métodos de seleção determinísticos. Um exemplo é a seleção por truncamento, que consiste em ordenar os indivíduos pela aptidão e escolher apenas os melhores para se reproduzirem. O trabalho de Pereira (2023) explica que esse tipo de seleção é chamado determinístico justamente por não envolver fatores aleatórios. Em contraste, métodos estocásticos, como a roleta, preservam a chance, ainda que pequena, de indivíduos menos aptos contribuírem para a próxima geração, o que pode ser vantajoso em problemas que exigem diversidade genética (AL-GHARAIBEH, 2007).

2.3.3 Recombinação

A recombinação, também conhecida como cruzamento (*crossover*), é um operador essencial nos Algoritmos Evolutivos, principalmente nos Algoritmos Genéticos. Sua função é combinar características de dois ou mais progenitores para gerar descendentes com potencial de se tornarem soluções mais adaptadas. Esse processo ocorre de forma estocástica, ou seja, guiado pelo acaso, mas sempre com o objetivo de aproveitar a informação existente nos pais para criar indivíduos com maior diversidade genética. Dessa forma, os descendentes herdam partes de cada progenitor, e essa mistura aumenta a chance de que novas combinações resultem em soluções de melhor qualidade.

Segundo Gabriel e Delbem (2008), a recombinação pode ser considerada o principal operador de reprodução nos Algoritmos Genéticos. Isto porque nos Algoritmos Genéticos é obrigatória a aplicação da recombinação para produção de um novo indivíduo, sendo a mutação um operador opcional.

Um exemplo bastante utilizado é a recombinação de 1-ponto. Neste caso, um corte é feito aleatoriamente no cromossomo dos progenitores, dividindo-os em duas partes. O descendente é, então, formado pela junção de uma parte do primeiro progenitor e outra parte do segundo. Esse mecanismo simples é suficiente para promover diversidade genética na população, evitando que todos os indivíduos sigam um mesmo padrão. A Figura 1

ilustra esse exemplo:

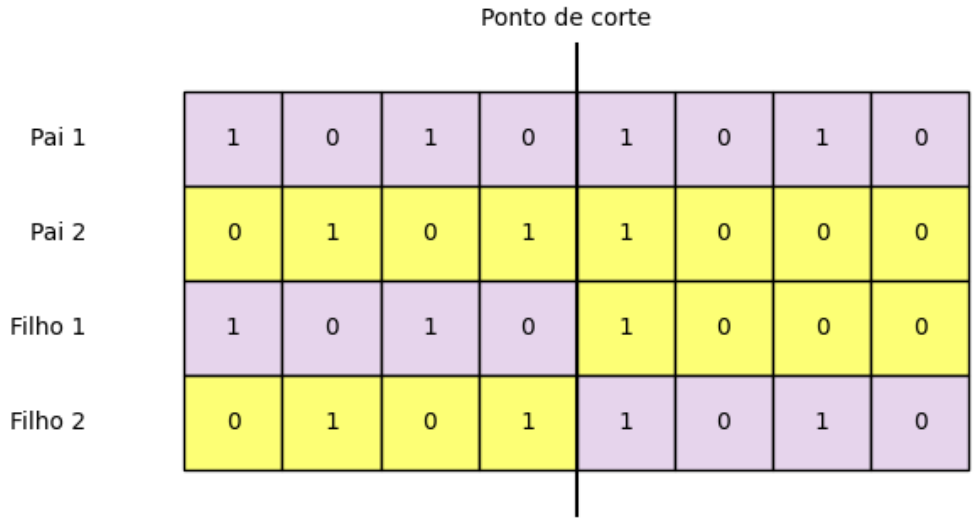

Figura 1 – Exemplo de um cruzamento de dois indivíduos.

Schwefel (1975) destaca que a recombinação pode acelerar a busca por soluções, pois facilita a adaptação dos parâmetros de variabilidade, como os desvios-padrão usados em mutações posteriores. Essa interação entre recombinação e mutação contribui para explorar o espaço de busca de forma mais eficiente, aumentando a probabilidade de alcançar soluções próximas do ótimo.

Gaspar-Cunha, Antunes e Takahashi (2012) descrevem ainda duas formas de recombinação muito utilizadas nas Estratégias Evolutivas. A primeira é a recombinação intermédia, em que os descendentes recebem valores que correspondem à média dos componentes de seus progenitores. Na recombinação discreta, cada componente do descendente é escolhido aleatoriamente a partir de um dos pais. Essas variações oferecem flexibilidade ao processo evolutivo, promovendo diversidade tanto em termos de exploração ampla quanto de ajustes refinados no espaço de soluções.

Holland (1992) compara a recombinação a um "teste de múltiplas combinações", em que diferentes arranjos de características são explorados na tentativa de identificar as associações mais vantajosas para a adaptação. Essa visão reforça o papel desse operador como um dos motores principais da inovação nos Algoritmos Evolutivos.

2.3.4 Mutação

A mutação é o operador responsável por introduzir uma pequena variabilidade genética nas populações, modificando aleatoriamente um ou mais genes de um cromossomo. Seu objetivo é gerar novos indivíduos (descendentes) que possam apresentar características diferentes e, potencialmente, mais vantajosas, em relação aos seus progenitores, sempre avaliados de acordo com uma função-objetivo.

Esse operador pode atuar de duas formas: criando uma cópia modificada de um indivíduo já existente ou aplicando alterações em descendentes gerados por recombinação. Goldberg (1989) explica que a mutação padrão consiste em trocar o valor de um gene em um cromossomo. Em uma codificação binária, por exemplo, um gene com valor 1 pode ser transformado em 0. Embora simples, esse mecanismo é crucial para que o algoritmo explore regiões ainda não visitadas do espaço de busca, evitando que a população fique restrita apenas à combinação dos genes já existentes.

Para exemplicar, tem-se na Figura 2 o progenitor gerando um descendente idêntico (clonagem), enquanto que na Figura 3 há uma alteração pontual no quarto gene. No caso binário, o bit é invertido; em representações reais, o valor é levemente perturbado por um ruído gaussiano. Essa modificação mínima já torna o descendente diferente do pai e abre uma nova vizinhança no espaço de busca, permitindo que o algoritmo investigue soluções que não seriam alcançadas apenas com a recombinação.

	0	1	2	3	4	5	6	7
Pai	0	1	0	1	0	1	0	1
Descendente	0	1	0	1	0	1	0	1

Figura 2 – Exemplo de um cromossomo sem mutação.

	0	1	2	3	**4**	5	6	7
Pai	0	1	0	1	**0**	1	0	1
Descendente	0	1	0	1	**1**	1	0	1

Figura 3 – Exemplo de um cromossomo com mutação.

A importância da mutação está justamente em impedir que o algoritmo fique preso em soluções locais. Ao introduzir pequenas alterações nos indivíduos, abre-se espaço para explorar novas regiões do espaço de soluções e, assim, alcançar desempenhos mais elevados. Al-Gharaibeh (2007) destaca que a taxa de mutação geralmente é baixa, pois seu papel é promover ajustes sutis em indivíduos já adaptados. Estas pequenas mudanças podem gerar descendentes capazes de superar os pais em termos de aptidão.

Portanto, o uso desse operador deve ser equilibrado. Uma taxa de mutação excessivamente alta pode comprometer a convergência do algoritmo, pois introduz variação demais e prejudica a preservação de boas soluções. Em contrapartida, uma taxa muito baixa pode gerar populações homogêneas, com pouca diversidade e baixa capacidade de adaptação. Assim, a mutação é um elemento que deve ser aplicado com cautela, mas cuja presença é indispensável para garantir que o processo evolutivo seja, de fato, dinâmico e eficaz.

2.3.5 Condição de parada

A condição de parada é o critério que define em que momento o processo de busca de um Algoritmo Evolutivo deve ser encerrado. Esse ponto é fundamental, pois sem ele o algoritmo poderia executar indefinidamente, consumindo recursos sem oferecer ganhos significativos. A escolha do critério mais adequado depende diretamente dos objetivos do estudo e da natureza do problema a ser resolvido.

A literatura aponta diferentes possibilidades para estabelecer esse limite. Entre elas estão: encerrar quando uma solução atinge um nível de aptidão considerado satisfatório; impor um tempo máximo de execução; definir uma quantidade limite de gerações; ou ainda interromper após um número pré-determinado de gerações consecutivas sem apresentar melhorias relevantes (GASPAR-CUNHA; ANTUNES; TAKAHASHI, 2012).

Com isto, conclui-se a apresentação dos principais elementos que compõem os Algoritmos Evolutivos. A seguir, será explorada outra técnica bioinspirada também utilizada nesta investigação: as Redes Neurais Artificiais.

Na próxima seção será abordada fundamentos da Rede Neural Artificial, as Redes Neurais Perceptron, Redes Neurais Multicamadas e toda fundamentação teórica.

2.4 Rede Neural Artificial

Antes de abordar o tema das Redes Neurais Artificiais, é necessário compreender o funcionamento do cérebro humano e de seus neurônios, o que será explorado nas subseções a seguir.

2.4.1 Funcionamento do Cérebro

O ponto de partida para o desenvolvimento das Redes Neurais Artificiais está no estudo do cérebro humano. O sistema nervoso é composto por células especializadas chamadas neurônios, que controlam o comportamento dos organismos vivos (NETO, 2017). Segundo o Sadava et al. (2009), cada neurônio possui três regiões principais:

- Axônio: é uma longa extensão do neurônio que conduz impulsos nervosos para outras células. Os axônios transmitem sinais elétricos do corpo celular para os dendritos de outros neurônios ou para outras células do corpo.
- Dendrito: são ramificações de entrada do neurônio que recebem sinais de outros neurônios. Eles conduzem esses sinais em direção ao corpo celular.
- Corpo Celular: também conhecido como somador, é a parte central do neurônio onde os sinais recebidos dos dendritos são somados. O corpo celular avalia a soma

dos estímulos e decide se o neurônio deve disparar um impulso elétrico em resposta aos sinais recebidos.

- Sinapse: é a junção entre os dendritos de um neurônio e os axônios de outros neurônios. As sinapses regulam a quantidade de informação que é transmitida entre neurônios e desempenham um papel crucial na memorização e armazenamento de informações.

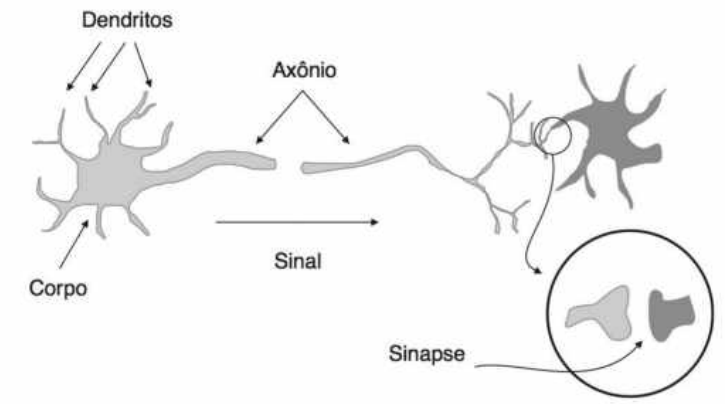

Figura 4 – Ilustração de um neurônio natural.

A Figura 4 foi retirada da obra de Faceli, Castro e Malabarba (2011) e apresenta a estrutura básica de um neurônio biológico. Essa representação é comumente utilizada como analogia para o desenvolvimento de redes neurais artificiais, destacando os principais componentes como os dendritos, o corpo celular e o axônio, que correspondem, respectivamente, às entradas, ao processamento e à saída de informações em um modelo computacional.

2.4.2 História do Neurônio Artificial

Em 1943, Warren McCulloch, psicólogo e neurofisiologista, e Walter Pitts, matemático, propuseram um modelo matemático para simular o comportamento dos neurônios biológicos. Neste modelo, cada informação de entrada é multiplicada por um peso e somada em um ponto de combinação linear. Se a soma resultante exceder um limiar determinado por uma função de ativação, uma resposta é gerada na saída do modelo (MCCULLOCH; PITTS, 1943). O modelo McCulloch-Pitts (MCP) simplifica o neurônio biológico, representando os dendritos como entradas de dados e o axônio como a saída. Os pesos sinápticos, positivos ou negativos, representam as sinapses, com pesos positivos indicando sinapses excitatórias e negativos indicando sinapses inibitórias (BRAGA; CARVALHO; LUDERMIR, 2007).

Inspirados nesse funcionamento biológico, Warren McCulloch e Walter Pitts propuseram, em 1943, o primeiro modelo matemático de um neurônio artificial para simular

o comportamento dos neurônios biológicos. Nesse modelo, cada informação de entrada recebe pesos numéricos e são combinadas linearmente. Se a soma ultrapassar um limiar definido por uma função de ativação, a saída é ativada; caso contrário, permanece inativa (LIMA; PINHEIRO; SANTOS, 2014). Esse modelo, conhecido como McCulloch-Pitts (MCP), representa de forma simplificada o funcionamento de um neurônio, sendo considerado o marco inicial do campo das redes neurais artificiais (BRAGA; CARVALHO; LUDERMIR, 2007).

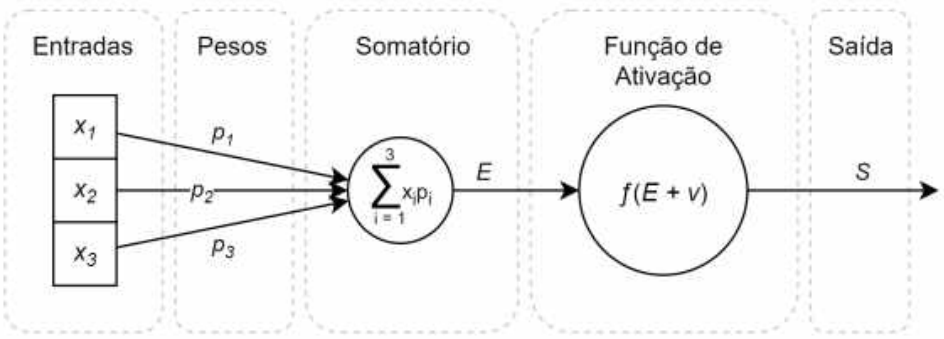

Figura 5 – Ilustração de um neurônio artificial por Pereira (2023).

A evolução da inteligência artificial, iniciada com modelos de neurônios artificiais propostos por McCulloch e Pitts em 1943, ilustra a contínua busca por máquinas capazes de simular comportamentos inteligentes (LIMA; PINHEIRO; SANTOS, 2014)

A seguir, será explicado o primeiro modelo de Rede Neural Artificial: o Perceptron.

2.4.3 Perceptron

Uma Rede Neural Artificial (RNA) é um modelo computacional inspirado no comportamento do sistema nervoso humano, com o objetivo de simular a capacidade do cérebro de aprender, adaptar-se, generalizar e organizar informações (LIMA; PINHEIRO; SANTOS, 2014). Assim como os neurônios biológicos, as RNAs são constituídas por unidades básicas de processamento, chamadas de neurônios ou nós, que estão interligadas, formando uma rede. Estas unidades estabelecem uma comunicação entre si por meio de conexões, onde cada ligação tem um peso específico, o qual é ajustado durante o processo de treinamento da rede. Este processo visa habilitar a RNA a aprender a partir de dados e, consequentemente, fazer inferências e generalizações sobre novos dados com base no conhecimento adquirido anteriormente (BRAGA; CARVALHO; LUDERMIR, 2007).

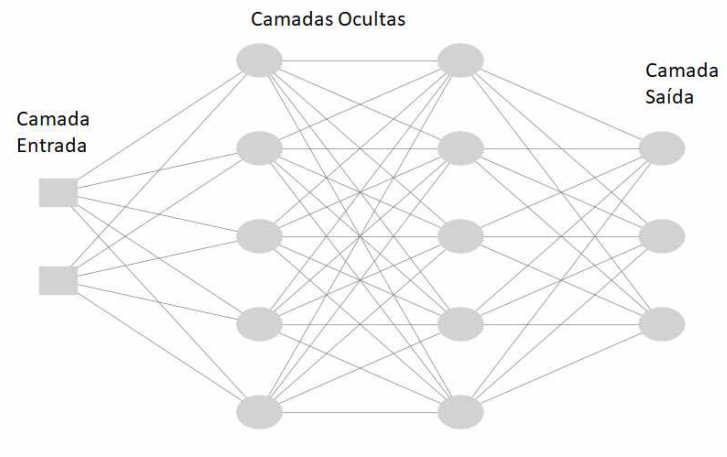

Figura 6 – Exemplo de uma Rede Neural Artificial.

Essas redes são conhecidas como modelos conexionistas, pois funcionam por meio de um conjunto de nós interligados que processam informações em paralelo. A capacidade de resolver problemas complexos está diretamente ligada ao processo de ajuste dos pesos, realizado por algoritmos de aprendizado. Entre eles, destaca-se o algoritmo de retropropagação (*backpropagation*), introduzido por Rumelhart e colaboradores em 1986, que marcou um avanço decisivo ao permitir o treinamento de redes com múltiplas camadas (RUMELHART; HINTON; WILLIAMS, 1986) e (LIMA; PINHEIRO; SANTOS, 2014).

Atualmente, as RNAs são aplicadas em diversas áreas, como reconhecimento de padrões, previsão de dados e aprendizado de máquina. Sua popularidade se deve à adaptabilidade, ao processamento paralelo e à capacidade de identificar padrões não lineares, características que as tornam ferramentas poderosas para lidar com problemas de alta complexidade (LIMA; PINHEIRO; SANTOS, 2014) e (BRAGA; CARVALHO; LUDERMIR, 2007).

Neste contexto, o Perceptron é um modelo fundamental na história das RNAs, introduzido por Frank Rosenblatt em 1958, sendo o primeiro modelo de Rede Neural Artificial implementado. Seu objetivo inicial foi enfrentar problemas de reconhecimento de padrões, algo natural para os seres humanos, mas extremamente desafiador para máquinas (BARRETO, 2002). Por definição, o Perceptron é composto por uma única camada de unidades de processamento, conhecidas como Unidades Lógicas com Limiar (LTU). Cada unidade recebe entradas ponderadas, aplica uma função de ativação e gera uma saída binária, resultante da soma dessas entradas (LIMA; PINHEIRO; SANTOS, 2014).

A função de ativação tem papel decisivo nesse processo, pois determina se o neurônio deve ou não ser ativado. Em outras palavras, ela transforma os sinais recebidos em um resultado que pode ser interpretado pela rede. Essa característica permitiu ao Perceptron realizar tarefas simples de classificação e lógica (BARRETO, 2002). Estas funções são

responsáveis por transformar o sinal de entrada em um resultado não-linear e determinar se um neurônio deve ser ativado ou não. Em termos simples, elas decidem se a saída do neurônio será ativada com base na entrada recebida, permitindo que a rede neural capture relações complexas e padrões nos dados. As funções de ativação podem ser lineares, diferenciáveis ou parcialmente diferenciáveis. As diferenciáveis são preferidas porque ajudam a mitigar problemas relacionados ao gradiente durante o treinamento da rede neural (LIMA; PINHEIRO; SANTOS, 2014)

Existem diversas funções de ativação comumente usadas, cada uma com suas próprias características e funcionamentos. A seguir, são apresentados alguns tipos de função de ativação e suas saídas:

- Função Linear: utilizada em modelos simples, mantém a proporcionalidade entre entrada e saída. Kohonen, em 1972, demonstrou seu uso em Mapas Auto-Organizáveis, permitindo que entradas, pesos e saídas fossem contínuos. (LIMA; PINHEIRO; SANTOS, 2014)

- Função Limiar: empregada nos primeiros modelos, como os de McCulloch e Pitts (1943) e Rosenblatt (1958), ativa a saída apenas quando a soma ponderada ultrapassa um limite pré-definido (LIMA; PINHEIRO; SANTOS, 2014). Essa função é frequentemente usada em neurônios binários, onde a saída é 1 se a entrada excede o limiar (limite definido), ou 0 caso contrário; ou seja, o neurônio é ativado quando a saída for 1 e desativado quando for zero.

- Função Sigmoidal: possui formato em S, limitando os valores de saída entre 0 e 1, o que suaviza as respostas (BRAGA; CARVALHO; LUDERMIR, 2007)

- Função Tangente Hiperbólica (tanh): semelhante à sigmoidal, mas com amplitude maior, gerando valores entre -1 e 1 (BRAGA; CARVALHO; LUDERMIR, 2007)

- Função *Rectified Linear Unit* (ReLU): combina características da função linear e da limiar, retornando 0 para entradas negativas e o valor da entrada para entradas positivas.

Em termos de aprendizado, o Perceptron ajusta seus pesos com base em um fator de aprendizado (N), que regula a velocidade com que os pesos são modificados durante o treinamento. Esse mecanismo permitia que o modelo se ajustasse a partir dos dados disponíveis e buscasse maior acurácia em classificações (PEREIRA, 2023).

Apesar disso, a limitação de não resolver problemas não linearmente separáveis motivou o surgimento de arquiteturas mais complexas. Nesse contexto, o Perceptron Multicamadas (MLP) surgiu como uma extensão do modelo básico, incorporando múltiplas

camadas ocultas de neurônios para lidar com padrões não lineares. As MLPs são conhecidas como *Feedforward Neural Networks* (FFNs) e, mesmo sendo uma das arquiteturas mais antigas desse tipo, continuam amplamente utilizadas (RAHMAN, 2024). Essa rede será detalhada na próxima subseção.

2.4.4 Perceptron Multicamadas

O Perceptron Multicamadas (MLP, do inglês *Multilayer* Perceptron) é uma evolução do modelo básico do Perceptron, introduzindo a ideia de múltiplas camadas de neurônios entre a entrada e a saída. Enquanto o Perceptron simples possui apenas uma camada de entrada conectada diretamente à camada de saída, a MLP inclui três elementos estruturais principais: a camada de entrada, uma ou mais camadas intermediárias (também chamadas de camadas ocultas) e a camada de saída (LIMA; PINHEIRO; SANTOS, 2014), como pode ser observado na Figura 7.

As camadas ocultas são as responsáveis por dar ao modelo maior poder de representação. Elas permitem que a rede capture padrões complexos e não lineares, processando as informações recebidas da camada de entrada e transmitindo os resultados para a próxima camada. Cada neurônio de uma camada está conectado a todos os neurônios da camada seguinte, sem ciclos ou conexões laterais, o que caracteriza a arquitetura como *feedforward* (BARRETO, 2002). Esse fluxo contínuo da informação é justamente o que fornece às MLPs a capacidade de superar as limitações do Perceptron, que só consegue lidar com problemas linearmente separáveis (LIMA; PINHEIRO; SANTOS, 2014) .

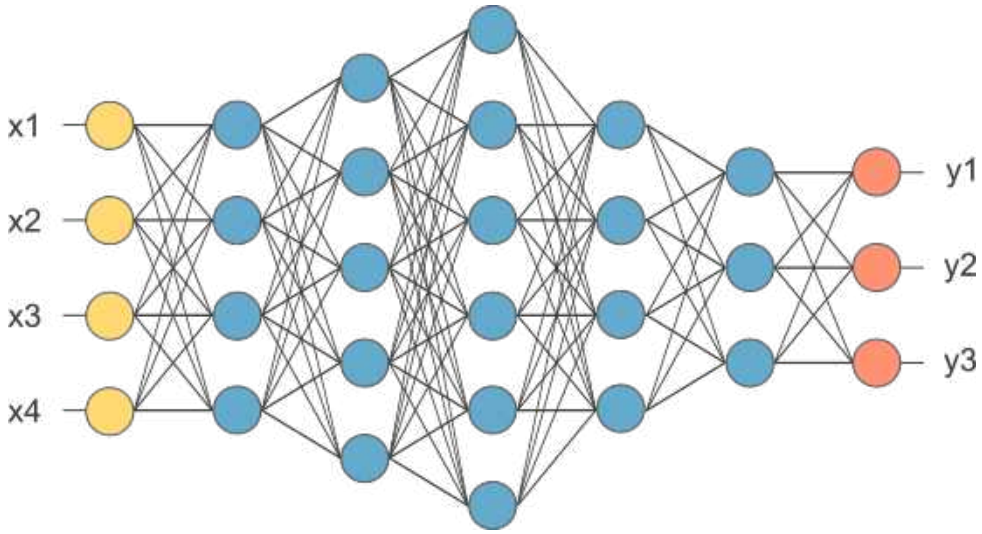

Figura 7 – Exemplo de uma Rede Neural Multicamadas (Laboratório iMobilis - UFOP, s.d.).

2.4.5 *Backpropagation*

Com a evolução das arquiteturas, surgiu a necessidade de métodos mais robustos de treinamento. Nesse contexto, o algoritmo de retropropagação dos erros (*backpropagation*) foi proposto por Rumelhart e colaboradores na década de 1980, tornando-se um divisor de águas no avanço das redes neurais (RUMELHART; HINTON; WILLIAMS, 1986).

O algoritmo parte de uma ideia simples: primeiro a rede recebe os dados de entrada e calcula uma saída. Em seguida, essa saída é comparada ao valor esperado e o erro resultante é calculado. Esse erro é, então, propagado de volta através da rede, camada por camada, ajustando os pesos das conexões de acordo com o gradiente descendente. O objetivo é reduzir progressivamente a diferença entre a saída prevista e a saída correta (RUMELHART; HINTON; WILLIAMS, 1986). Sua eficácia fez do *backpropagation* uma das técnicas mais populares para o treinamento de redes neurais, especialmente em arquiteturas profundas, tornando viável a utilização de MLPs para resolver problemas que não são linearmente separáveis, ampliando significativamente o campo de aplicação das redes neurais (LIMA; PINHEIRO; SANTOS, 2014).

2.4.6 Treinamento e teste

O treinamento de uma rede neural é o processo de ajuste dos pesos e vieses com o objetivo de reduzir o erro entre a saída prevista e a saída desejada. Nessa fase, a rede é exposta a um conjunto de exemplos rotulados, os dados de treinamento, e aprende padrões e associações entre entradas e alvos. Os algoritmos de aprendizagem, como a retropropagação, definem como e quando atualizar os parâmetros internos para minimizar o erro de previsão (LIMA; PINHEIRO; SANTOS, 2014).

O teste, por sua vez, avalia a capacidade de generalização da rede em dados inéditos, não utilizados no treinamento. Essa etapa verifica se o modelo transfere o que aprendeu para situações novas, medindo acurácia e desempenho fora da amostra de treino.

Do ponto de vista conceitual, distinguem-se três ideias que se complementam. O aprendizado é o processo pelo qual a rede adquire conhecimento a partir dos dados rotulados. O treinamento é a execução prática desse processo, isto é, o ajuste efetivo de pesos e vieses sobre o conjunto de treino. A generalização é a habilidade de aplicar o conhecimento adquirido a exemplos não vistos, sendo aferida no momento do teste.

Quanto aos paradigmas, diferentes formas de aprendizado podem ser empregadas:

- Aprendizado supervisionado: utiliza pares entrada–saída. As previsões da rede são comparadas com os rótulos corretos (gabarito) e o erro guia os ajustes (LIMA; PINHEIRO; SANTOS, 2014).
- Aprendizado não supervisionado: trabalha apenas com entradas, buscando descobrir

estruturas, padrões e agrupamentos de forma autônoma. Entre as regras clássicas está a de Hebb, que reforça conexões quando ativações ocorrem conjuntamente (LIMA; PINHEIRO; SANTOS, 2014).

- Aprendizado por reforço: envolve interação com um ambiente, com feedback em forma de recompensas e punições. O objetivo é maximizar a recompensa acumulada por meio de estratégias de ação, como no Q-Learning e em suas variantes profundas (LIMA; PINHEIRO; SANTOS, 2014).

Antes do treinamento, algumas práticas ajudam a obter modelos mais estáveis e com boa generalização:

- Separação dos dados: cria conjuntos distintos de treinamento e validação, de modo a acompanhar o desempenho fora do treino durante o ajuste. As proporções usuais incluem 70% ou 80% para treino e 30% ou 20% para validação.
- Conjunto de treinamento: é utilizado para atualização dos parâmetros.
- Conjunto de validação: é usado para monitorar desempenho em dados não vistos durante o ajuste, auxiliando na detecção de sobreajuste.
- Inicialização de pesos e vieses: as escolhas adequadas de inicialização favorecem a convergência e evitam treinos excessivamente lentos ou instáveis.

Finalmente, a capacidade de generalização refere-se à habilidade da rede neural de aplicar o conhecimento adquirido durante o treinamento a novos dados que não foram vistos anteriormente. Em outras palavras, uma rede bem generalizada pode responder de forma adequada a padrões que não estavam presentes no conjunto de treinamento.

2.4.7 Sobreajuste

Durante o treinamento, os pesos são atualizados com o objetivo de reduzir o erro observado nos exemplos de treino. Caso o processo se estenda por tempo excessivo ou sem controle, pode surgir o sobreajuste (*overfitting*) (SRIVASTAVA et al., 2014). Nessa condição, o modelo passa a memorizar particularidades do conjunto de treino em vez de capturar regularidades gerais, o que leva a bom desempenho apenas nos dados vistos e queda de desempenho em dados novos.

O sobreajuste é identificado quando o erro no conjunto de validação começa a aumentar, enquanto o erro no treino continua diminuindo. Esse comportamento indica que o modelo está se especializando nos detalhes do treino e perdendo capacidade de generalização.

Algumas práticas ajudam a conter esse efeito. A validação cruzada permite acompanhar o desempenho fora do treino durante o processo de ajuste. A técnica de *early stopping* (PRECHELT, 1998), ou parada antecipada, interrompe o treinamento quando o desempenho no conjunto de validação deixa de melhorar por um número pré-definido de ciclos, preservando o ponto em que a rede apresentou melhor generalização. Outra técnica importante é a *dropout* (SRIVASTAVA et al., 2014), que consiste em desligar aleatoriamente (ou seja, zerar) uma fração dos neurônios da rede durante o treinamento. Isso força a rede a não depender demais de nenhum neurônio específico, o que ajuda a melhorar sua capacidade de generalização. Essas estratégias mantêm o equilíbrio entre reduzir o erro de treinamento e conservar a capacidade do modelo de atuar bem em dados inéditos.

2.5 Considerações finais

Este capítulo reuniu os pilares conceituais que sustentaram o desenvolvimento deste Trabalho de Conclusão de Curso. Primeiramente, foram apresentados fundamentos de otimização computacional e heurísticas. Em seguida, foi possível detalhar a estrutura dos Algoritmos Evolutivos, bem como seus componentes canônicos: *fitness*, seleção, recombinação, mutação e critérios de parada, incluindo variações de estratégias evolutivas que modulam pressão seletiva e diversidade. A seguir, apresentou-se o panorama das Redes Neurais Artificiais desde a inspiração biológica até os modelos clássicos, com ênfase no Perceptron, nas Redes Neurais Multicamadas e no treinamento por retropropagação, além de noções práticas de treinamento, teste e riscos de sobreajuste. Em conjunto, esses tópicos compuseram a fundamentação teórica que direcionou a elaboração deste trabalho.

3 Trabalhos correlatos

Este capítulo abordará trabalhos que iteragem diretamente com o uso de técnicas de inteligência artificial em jogos digitais, com ênfase em Redes Neurais Artificiais e Algoritmos Evolutivos. A seleção contempla um panorama introdutório, uma aplicação em jogos baseados em turno e, principalmente, o trabalho que fundamenta a metodologia adotada neste estudo.

3.1 Braga, Carvalho e Ludermir (2007)

A obra *Redes Neurais Artificiais: Teoria e Aplicações* (BRAGA; CARVALHO; LUDERMIR, 2007) é uma referência consolidada para fundamentos de RNAs, cobrindo aprendizado supervisionado e não supervisionado, arquiteturas e algoritmos de treinamento. Além de sustentar a fundamentação teórica, o livro oferece conexões práticas que justificam escolhas metodológicas no desenvolvimento de agentes inteligentes em jogos.

3.2 Costa, Prampero e Salazar (2009)

Este material didático da UNICAMP (COSTA; PRAMPERO; SALAZAR, 2009) oferece uma introdução panorâmica à IA em jogos, organizando conceitos, histórico e ramificações da área. Esta obra diferencia "IA acadêmica" de "IA para jogos" como abordagens complementares e apresenta técnicas recorrentes na indústria, como padrões de comportamento, máquinas de estados e redes neurais. Como referência de base, o texto legitima o uso de RNAs em jogos e destaca a importância de soluções pragmáticas orientadas à experiência do jogador, servindo de alicerce conceitual para estudos aplicados.

3.3 Oliveira (2018)

O trabalho de Oliveira (2018) explora métodos de IA em jogos baseados em turnos, como o MiniDungeons, combinando aprendizado por reforço e algoritmos evolutivos para agentes que tomam decisões estratégicas em ambientes 2D com monstros e tesouros. Os resultados reforçam a viabilidade de Algoritmos Evolutivos para melhorar a performance de agentes e para analisar o comportamento de jogadores. Embora o gênero seja distinto do jogo do dinossauro, a pesquisa é relevante por evidenciar que princípios de IA, incluindo redes neurais e evolução, transitam entre gêneros mediante ajustes de representação de estado e objetivo.

3.4 Pereira (2023)

O estudo de Pereira (2023) investigou o uso de Estratégias Evolutivas no treinamento de Redes Neurais Artificiais aplicadas a jogos digitais. Entre os ambientes de teste, o autor recriou o jogo do dinossauro do Google, no qual o agente controlado por IA deveria aprender a evitar obstáculos de forma autônoma. A arquitetura adotada seguiu o modelo de RNA Multicamadas, com seis entradas, uma camada oculta com doze neurônios e duas saídas correspondentes às ações de pular e abaixar. O treinamento ocorreu por mutações aleatórias nos pesos e seleção dos indivíduos mais aptos, medidos pela pontuação associada ao tempo de sobrevivência no jogo.

Este estudo é a principal referência metodológica para o presente trabalho, por demonstrar a viabilidade de combinar RNA com AE em um ambiente de corrida infinita. Em particular, a configuração de sensores e a forma de avaliação via sobrevivência constituem um ponto de partida sólido para ajustes arquiteturais e de critério de *fitness*.

3.5 Síntese comparativa

Os trabalhos analisados convergem em dois pontos centrais. Primeiro, as RNAs são adequadas para modelar comportamentos em jogos, desde soluções mais clássicas até cenários interativos complexos (COSTA; PRAMPERO; SALAZAR, 2009; BRAGA; CARVALHO; LUDERMIR, 2007). Segundo, os métodos evolutivos são eficazes para treinar agentes quando se busca explorar o espaço de soluções sem depender de gradientes, seja em turnos (OLIVEIRA, 2018) ou em corrida infinita (PEREIRA, 2023).

Em relação a Pereira (2023), que constitui a base metodológica, este estudo se posiciona no mesmo gênero de jogo e aprofunda a discussão em três eixos:

- Representação de entrada e objetivo: enquanto o trabalho de Pereira utiliza seis entradas e *fitness* por sobrevivência, a proposta atual trabalhou com três entradas específicas do ambiente e *fitness* orientado a ações corretas, aproximando a função de avaliação do comportamento desejado.
- Arquitetura: em vez de uma única camada oculta com doze neurônios, como no trabalho de Pereira, adotou-se uma arquitetura com duas camadas ocultas de seis neurônios cada, buscando equilíbrio entre capacidade de representação e custo computacional.
- Operadores evolutivos: além de mutação, empregou-se seleção por roleta e recombinação, investigando como diferentes operadores influenciam a exploração do espaço do aprendizado.

Com esse recorte, a contribuição deste trabalho não residiu apenas em reproduzir o cenário feito por Pereira (2023), mas em comparar escolhas arquiteturais e de avaliação mais alinhadas ao comportamento alvo no jogo do dinossauro, oferecendo evidências sobre como sensores, função de *fitness* e operadores evolutivos impactam o desempenho do agente.

3.6 Considerações finais

O levantamento de trabalhos relacionados ao tema confirmou que as RNAs e os AEs são estratégias pertinentes para agentes em jogos digitais, e que diferentes escolhas de sensores, função de avaliação e operadores de evolução alteram substancialmente o desempenho. Em particular, os estudos analisados validaram a viabilidade do cenário de corrida infinita e indicaram lacunas que foram exploradas nesta pesquisa, tais como: o uso de seleção por roleta combinada a recombinação e mutação, e análise sistemática de parâmetros como a taxa de mutação e o tamanho populacional.

4 Desenvolvimento

Este capítulo apresentará o desenvolvimento deste Trabalho de Conclusão de Curso, com ênfase na implementação da Rede Neural Artificial e do Algoritmo Evolutivo, ambos aplicados ao jogo do Dinossauro.

4.1 Recursos de desenvolvimento

O trabalho foi implementado e testado em ambiente local com as seguintes especificações: processador Intel® Pentium® 5405U @ 2,30 GHz, memória RAM 12 GB, sistema operacional Windows 10, 64 bits, utilizando Java Development Kit – Oracle OpenJDK 22.0.2 e o ambiente de desenvolvimento IntelliJ IDEA Community Edition 2024.2.1.

Essas configurações mostraram-se suficientes para a implementação do jogo, da rede neural e do algoritmo evolutivo, bem como para a execução dos experimentos. Toda a solução foi construída com bibliotecas padrão da linguagem, sem dependência de *frameworks* externos.

4.2 Referência principal para a implementação do jogo

O jogo do dinossauro do *Google Chrome* é um *runner* infinito amplamente conhecido e adequado para experimentos controlados, que foi utilizado por Pereira (2023).

A Figura 8 apresenta uma ilustração do jogo original.

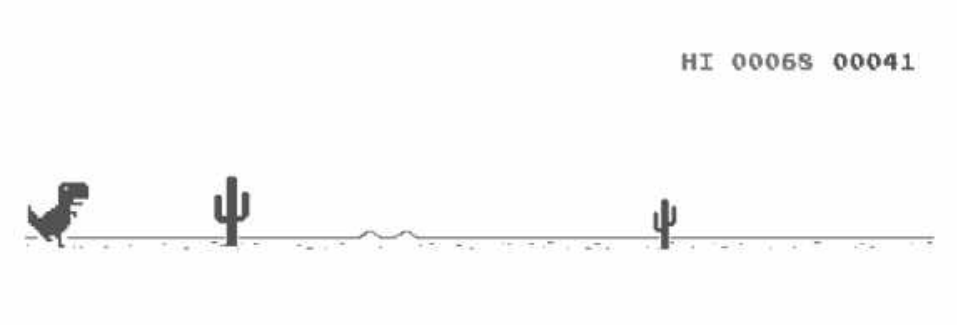

Figura 8 – Captura do jogo implementado por Pereira (2023).

O estudo de Pereira (2023) foi a referência inicial, especialmente pela adaptação do jogo do dinossauro e pela aplicação de RNAs em um ambiente interativo. Embora o seu trabalho contemple outros jogos, o presente estudo concentrou-se exclusivamente no jogo do dinossauro do Google e aprofundou a análise sobre como diferentes escolhas de arquitetura e de operadores evolutivos impactam o desempenho do agente. A relação

com a implementação de Pereira (2023) foi constante, embora os objetivos e os recortes experimentais adotados neste estudo foram distintos.

Quanto à implementação, o trabalho de Pereira (2023) foi desenvolvido em linguagem C com apoio de bibliotecas gráficas externas. Neste estudo, foi adotado Java para todo o desenvolvimento, desde a interface gráfica até a lógica do jogo e os módulos de aprendizagem. A decisão privilegia homogeneidade tecnológica, portabilidade e reprodutibilidade do experimento, permitindo a execução do projeto com a instalação padrão de Java e garantindo integração nativa entre os componentes do jogo, da RNA e do AE.

O interesse principal deste trabalho foi analisar o comportamento aprendido pelos dinossauros diante de variações controladas de arquitetura de rede e de parâmetros evolutivos. Para tal, o código foi estruturado com atenção à organização, à legibilidade e a boas práticas de programação, favorecendo a inspeção, a manutenção e a replicação dos testes.

Na seção seguinte, será apresentada a organização em que os códigos foram dispostos, de modo a facilitar a sua manipulação e seu entendimento.

4.3 Organização da implementação do sistema

Esta etapa organiza a aplicação em camadas bem definidas para isolar responsabilidades e facilitar manutenção. A interface gráfica foi construída em Java com a biblioteca `javax.swing`, que disponibiliza componentes portáveis e consistentes entre plataformas. Entre esses componentes, destaca-se o `JPanel`, empregado como superfície de desenho e contêiner de elementos visuais.

Foram definidas as classes `GameWindow` e `GamePanel`, seguindo o princípio de separação de responsabilidades. A `GameWindow` configura e exibe a janela do jogo, enquanto a `GamePanel` é responsável pelo ciclo de renderização e pela apresentação dos elementos na tela (Figura 9). Essa divisão reduziu o acoplamento entre criação de janela e desenho, além de simplificar testes e futuras extensões.

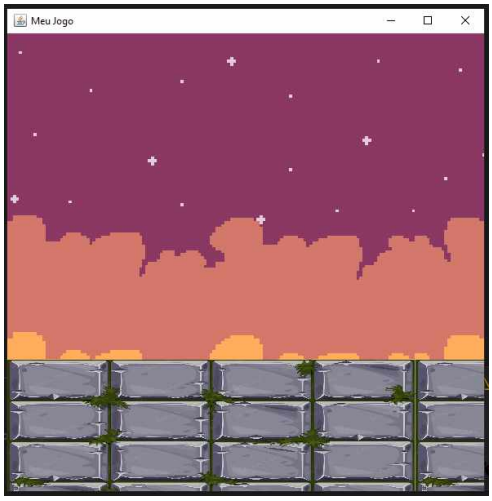

Figura 9 – Captura de tela 600X600.

Com a infraestrutura gráfica pronta, os componentes centrais da mecânica foram organizados de forma modular:

- **Movimento**: controla deslocamentos, saltos, agachamento e aplicação da gravidade, mantendo a atualização de posição em função do tempo.
- **Sprite**: gerencia carregamento, animação e troca de imagens de personagens e objetos.
- **Sensores**: realiza detecção de colisões e interações com o ambiente, expondo eventos para a lógica do jogo.
- **Som**: reproduz efeitos sonoros pontuais, mantendo a possibilidade de expansão futura do *feedback* audiovisual.

Figura 10 – Cenário do jogo implementado.

A partir dessa base, os agentes foram modelados pela classe `PlayerIA`. Cada dinossauro representou uma instância capaz de integrar percepção, decisão e ação, coordenando os módulos de movimento, sprites, sensores e som. A `PlayerIA` operou como ponto de encontro entre a lógica do jogo e os mecanismos de aprendizado, servindo ao mesmo tempo como entidade jogável e como indivíduo dentro do processo evolutivo.

Nas próximas seções, serão descritos detalhes de implementação dos dinossauros e inimigos, da RNA, do AE, e a interação da rede, dinossauro e cenário.

4.4 Implementação do dinossauro e do inimigo

O primeiro objetivo prático foi recriar o jogo do dinossauro preservando suas mecânicas essenciais. Para tal, o personagem foi implementado com a capacidade de correr em um cenário contínuo enquanto obstáculos surgiam de forma aleatória e procedural, exigindo respostas rápidas para evitar colisões. Foram considerados dois tipos de inimigos, em alinhamento com o comportamento clássico do jogo: cactos, que demandavam salto, e pterossauros, que exigiam abaixar.

O esquema de controle foi mantido simples e direto. A tecla *Espaço* acionava o salto e a tecla *S* acionava o abaixar, refletindo as duas ações válidas do personagem no eixo vertical como na Figura 11. No estudo de Pereira (2023), por exemplo, há um experimento com um avião para sobrevoar áreas com espinhos; neste estudo, o conjunto de inimigos foi limitado a terrestres e voadores, sem a inclusão de novos obstáculos ou mecânicas auxiliares de voo.

Dinossauro em movimento. | Dinossauro pulando. | Dinossauro abaixando.

Figura 11 – Estados do dinossauro durante a interação com obstáculos.

A dinâmica do salto foi determinada pela força da gravidade. O personagem foi programado para saltar apenas quando em contato com o solo, impedindo múltiplos saltos consecutivos no ar. Essa restrição contribuiu para um comportamento mais realista e previsível, além de reduzir decisões ambíguas por parte da rede neural. Simultaneamente, a velocidade de deslocamento dos inimigos foi projetada para aumentar progressivamente ao longo da execução, elevando gradualmente o nível de desafio. Essa estratégia atuou como um currículo de dificuldade: no início, o ambiente apresentava-se mais lento e tolerante; com o tempo, tornava-se mais rápido e exigia respostas mais precisas por parte dos agentes.

A criação dos inimigos (Figura 12) foi baseada em um gerador procedural, no qual novos obstáculos eram instanciados na borda direita da tela e se deslocavam linearmente até desaparecerem do campo de visão. A cada *tick* do jogo, um contador interno era incrementado. Considerando um atraso médio de aproximadamente 16 milissegundos por quadro, o sistema operava em torno de 60 quadros por segundo. Esse contador, referido nesta pesquisa como cronômetro, coordenava dois comportamentos principais: a geração de um novo inimigo a cada 100 quadros e o aumento gradual da velocidade global em intervalos predefinidos. Assim, o cronômetro funcionava como um gatilho tanto para a geração dos inimigos quanto para a progressão da dificuldade do jogo.

Figura 12 – Inimigos do jogo original.

A colisão foi tratada por detecção de interseção entre áreas retangulares associadas aos elementos do jogo. Quando a área do personagem se sobrepunnha à área de um inimigo, ocorria uma eliminação imediata do indivíduo. O mesmo mecanismo restringiu a passagem pelas bordas e manteve o contato correto com o chão. A precisão dessa detecção foi essencial para que o *feedback* ao agente fosse consistente, evitando falsos positivos ou negativos que comprometeriam o aprendizado.

Do ponto de vista de avaliação, duas informações foram centrais: ***fitness*** e cronômetro. O *fitness* media a qualidade do comportamento do agente com base em ações corretas executadas ao longo do episódio. Deste modo, os acertos incrementavam o *fitness*; decisões tardias ou incorretas tendiam a resultar em colisão e eliminação. Quando um indivíduo era eliminado, pesos atuais da rede, valor de *fitness* e marca do cronômetro eram armazenados para análise. Em termos intuitivos, os valores altos de *fitness* indicavam sequência consistente de decisões adequadas.

Cada execução definia dois parâmetros globais: o número de gerações e o tamanho da população. Nos experimentos de referência, utilizou-se 30 gerações e população de 20 indivíduos. A primeira geração tinha pesos aleatórios e, como esperado, apresentava desempenho inferior. Com o avanço das gerações, o processo evolutivo selecionava e combinava soluções promissoras, elevando a capacidade dos agentes de lidar com os obstáculos.

Por fim, cada agente foi instanciado pela classe `PlayerIA`, que integrou percepção, decisão e ação. Essa classe conectou a lógica do jogo aos módulos de aprendizado, permitindo observar em tempo real a evolução do comportamento na interface gráfica. As seções seguintes detalham a implementação do Algoritmo Evolutivo e da Rede Neural Artificial, bem como a forma de interação entre rede, dinossauro, inimigos e cenário.

4.5 Implementação do Algoritmo Evolutivo

O treinamento ocorreu durante a execução do AE, em que a cada nova execução era criada uma população de indivíduos, todos inseridos no ambiente do *runner*. Os obstáculos apareceriam de forma procedural e aleatória, garantindo variedade de situações. Cada dinossauro (indivíduo) acumulava o *fitness* ao executar a ação correta no tempo adequado. As colisões resultavam em eliminação imediata e registro dos dados do indivíduo, incluindo pesos da rede e pontuação final.

O ciclo de uma geração seguiu as etapas a seguir:

1. **Inicialização**: criação de N indivíduos com pesos e vieses aleatórios.
2. **Avaliação**: execução no ambiente; ações corretas incrementavam o *fitness*; colisão eliminavam o indivíduo.
3. **Seleção**: ordenação por *fitness* e aplicação do método da roleta para escolha de pares de progenitores, proporcionalmente ao desempenho.
4. **Reprodução**: geração de descendentes por recombinação dos pesos dos pais.
5. **Mutação**: aplicação de taxa de mutação de 10% nos descendentes, introduzindo perturbações controladas para preservar diversidade.
6. **Elitismo**: preservação de 20% do topo da geração atual sem alterações, garantindo que soluções promissoras não se percam.

Este processo era repetido até que o critério de parada definido fosse atingido.

4.5.1 Representação dos indivíduos

Neste estudo, o indivíduo representou um dinossauro do jogo do *Google Chrome*. No jogo clássico, existe apenas um exemplar de dinossauro. Para este trabalho específico, implementou-se o Algoritmo Evolutivo para gerenciar uma população de N dinossauros (sendo N o tamanho da população), onde cada um deles tinha um valor de aptidão (*fitness*), em que se desejava maximizar.

Uma população inicial era gerada com uma lista de N dinossauros em que cada um possuía a sua própria Rede Neural Artificial, preliminarmente construída com pesos aleatórios. O objetivo desta rede neural foi identificar as ameaças e classificar qual ação o indivíduo deveria efetuar (pular/abaixar), visando evitar os obstáculos.

Cada dinossauro trabalhava com um campo de visão, a fim de que sempre que uma ameaça entrasse nessa região, a rede neural fosse capaz de detectar e classificá-la com base em suas coordenadas.

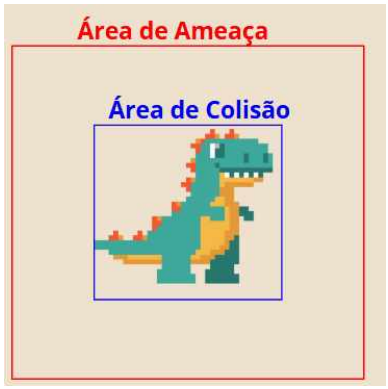

Figura 13 – Exemplo de um campo de visão do dinossauro.

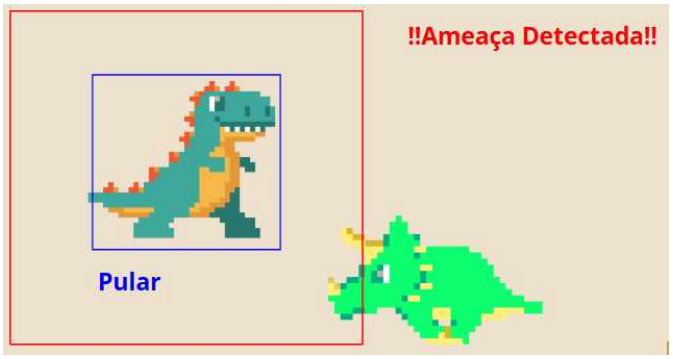

Figura 14 – Exemplo de uma ameaça detectada.

Como havia uma população de dinossauros jogando ao mesmo tempo, o campo de visão não reconhecia outros dinossauros como ameaça. Assim, mesmo que dois dinossauros estivessem próximos, não seriam tratados como ameaças entre si.

Quando um indivíduo colidia com um obstáculo, era eliminado e registravam-se o tempo de vida, o *fitness* e os pesos da rede. Após a eliminação de toda a população, aplicava-se o ciclo evolutivo — seleção, cruzamento, mutação e elitismo — para formar a nova geração. O processo era repetido até o critério de parada definido. Os parâmetros evolutivos são detalhados nas seções seguintes.

4.5.2 Função de aptidão

Em cada geração, o *fitness* de cada dinossauro media sua aptidão: enquanto houvesse ao menos um inimigo dentro do "radar" (campo de visão), o indivíduo recebia +1 ponto por quadro. Como ações incorretas resultavam em colisão imediata, apenas os indivíduos que decidiam corretamente prolongavam a sobrevivência e, consequentemente, acumulavam mais *fitness*. Quando ocorria colisão, o indivíduo era eliminado e a contagem de *fitness* era encerrada.

As entradas da rede neural, atualizadas a cada quadro, foram: distância horizontal $\Delta x =$ `inimigo.getX()` − `playerIA.getX()`, diferença de altura $\Delta y =$ `inimigo.getY()` − `playerIA.getY()` e velocidade global dos inimigos. A rede decidia entre duas ações: pular(saída[0] $>$ 0) ou abaixar(saída[1] $>$ 0).

Como a velocidade aumentava ao longo da geração, os encontros tornavam-se mais difíceis; logo, manter-se vivo por mais tempo em velocidades elevadas geralmente resultava em *fitness* maior.

Portanto, a função de aptidão premiava a sobrevivência nas ações realizadas corretamente, penalizava erros com eliminação imediata e escalava com a dificuldade crescente do jogo.

4.5.3 Seleção

A seleção definia como os indivíduos-pais seriam escolhidos a partir do desempenho observado no jogo. Foram implementados dois esquemas: torneio de N e roleta. Neste estudo, foram testados N=2 e N=3. A roleta, por sua vez, é o método adotado como padrão e, nesta etapa, não é alterado.

- Torneio de N: Em cada torneio, amostravam-se N indivíduos da população sem reposição dentro do torneio. Comparava-se o *fitness* e o melhor venceria, sendo escolhido como pai. Para definir o segundo pai, realizava-se um novo torneio com outra amostra de N candidatos. O procedimento era repetido até formar todos os pares necessários.
- Roleta: Todos os indivíduos concorrem em uma roleta virtual na qual a fatia de cada um é proporcional ao seu *fitness*. O sorteio selecionava um primeiro pai e, em seguida, um segundo, mantendo-se a regra de não reposição dentro do par para impedir duplicidade do mesmo progenitor. O processo era repetido até completar o conjunto de pais. Essa estratégia favorecia os mais aptos, mas preservava chance não nula para indivíduos com *fitness* menor, o que ajudaria a manter diversidade.

Experimentalmente, a seleção por roleta apresentou resultados mais consistentes e, por isso, foi mantida como método padrão nas demais análises.

4.5.4 Cruzamento

No cruzamento (ou *crossover*), a partir de um par de progenitores, o descendente seria gerado por média elemento a elemento entre os DNAs, mantendo o alinhamento de índices (mesma posição corresponde ao mesmo peso ou viés em ambas as redes). Por exemplo:

- **Pai 1:** [10,20,30]
- **Pai 2:** [6,26,24]
- **Filho:** [(10+6)/2,(20+26)/2,(30+24)/2]=[8,23,27].

Na prática, o filho tendia a herdar um comportamento intermediário entre as estratégias dos pais. No código, essa rotina está implementada em `crossover(p1, p2)`.

4.5.5 Mutação

Na mutação, após o cruzamento, podia ser aplicada uma variação controlada aos genes para introduzir novidade genética. Nesta implementação, cada posição do DNA

tinha chance de receber uma pequena perturbação aditiva, produzindo ajustes finos de 10% sobre o valor, como 23→ 25,3 ou 20,7. Esse perturbação evitaria que a população se tornasse homogênea e favoreceria a descoberta incremental de combinações de pesos mais eficazes para decidir entre pular e abaixar. A mutação foi aplicada pela função `aplicarMutacao()`.

4.5.6 Elitismo

Neste trabalho, o elitismo foi usado com taxa de 20%, isto é, 20% do topo da população era copiado tal como estava para a geração seguinte. Esse mecanismo impedia regressões acidentais e acelerava a consolidação de comportamentos que se mostraram robustos no jogo.

Nesta implementação, o elitismo não foi elaborado como uma função isolada, mas um parâmetro explícito que definia quantos indivíduos seriam preservados. O número de indivíduos de elite foi calculado multiplicando-se o tamanho da população (`numPlayers`) pela fração desejada (X), resultando em `int numElite = numPlayers*X;`. Por exemplo, para uma população de 100 indivíduos e elitismo de 20%, tem-se $X = 0.2$, logo `numElite = 20`. Esses `numElite` indivíduos são clonados diretamente para a nova geração, sem aplicação de mutação.

Em conjunto, estes três operadores cumpriram papéis complementares ao processo evolutivo do algoritmo: o cruzamento combinando padrões existentes, a mutação acrescentando variação local e o elitismo assegurando a continuidade do que já se mostrou eficaz. No contexto do *runner*, esse mecanismo resultou em descendentes que refinaram progressivamente o tempo de reação frente aos inimigos terrestres e voadores, enquanto a população como um todo manteve diversidade suficiente para se adaptar ao aumento gradual da velocidade dos inimigos.

4.5.7 Critério de parada

O processo evolutivo utilizou critério de parada global por número fixo de gerações. Nos experimentos de referência, `totalGeracao` foi definido como 30. Ao alcançar a 30ª geração, a simulação era finalizada, mantendo previsibilidade de tempo e comparabilidade entre execuções com o mesmo orçamento evolutivo.

Dentro de cada geração, a execução prosseguia até que todos os dinossauros fossem eliminados, isto é, até `quantidadeVivos` <= 0. Esse evento encerrava apenas a rodada corrente e não o processo global. No fechamento da geração eram realizadas as etapas a seguir:

- Coleta dos desempenhos individuais (*fitness* e demais registros);

- Seleção dos pais por roleta com elitismo conforme parametrização;
- Criação da nova população por cruzamento e aplicação de mutação;
- Reinicialização dos contadores e estados necessários para a próxima rodada;
- Incremento de `geracaoAtual`.

Há alternativas de parada possíveis, como alvo mínimo de *fitness*, janela de paciência sem melhora ou limite de tempo de execução. Nesta investigação, o critério oficial permaneceu o encerramento após alcançar um número pré-definido de gerações.

4.6 Implementação da Rede Neural Artificial

A RNA adotada seguiu o modelo multicamadas, estruturada em três blocos: uma camada de entrada que captava o estado do ambiente, duas camadas ocultas responsáveis pelo processamento não linear e uma camada de saída que indicava a ação do agente. A configuração foi desenhada para o cenário específico de corrida infinita com dois tipos de inimigos, terrestres e voadores, cujo enfrentamento exige decisões rápidas e binárias.

A camada de entrada possuía três neurônios, representando informações minimais porém suficientes para a tomada de decisão imediata:

Distância horizontal (inimigo.getX() - playerIA.getX()): media o quanto o inimigo está próximo do player no sentido horizontal, ou seja, estimava o tempo restante até o encontro, orientando a antecedência da ação.

Diferença de altura (inimigo.getY() - playerIA.getY()): identificava se o inimigo estava no solo ou no ar, permitindo que a IA previsse se devia pular ou se abaixar para evitar a colisão.

Velocidade dos inimigos (velocidadeInimigos): informava a rapidez com que o obstáculo se aproximava. Com isso, a rede conseguia calcular se precisava agir imediatamente ou se podia esperar um pouco mais antes de tomar a decisão.

Essas variáveis compunham um campo de visão simplificado. Em termos práticos, elas indicavam o que viria pela frente, onde estava e em que velocidade se aproximava. As duas camadas ocultas possuiam seis neurônios cada, oferecendo capacidade de representação suficiente para capturar relações não lineares sem onerar o custo computacional. A camada de saída continha dois neurônios, associados às ações disponíveis: pular e abaixar. A decisão era tomada a partir das ativações de saída, mapeadas diretamente para os comandos do agente.

Para acoplar percepção e ação, foi implementado um mecanismo de campo de visão que registrava coordenadas do inimigo assim que este entrava em faixa útil de detecção.

A partir desse ponto, as três entradas eram atualizadas a cada quadro e propagadas pela rede, permitindo que o agente antecipasse a resposta antes do contato.

Os testes iniciais com um único indivíduo evidenciaram comportamento altamente estocástico devido à inicialização aleatória de pesos e vieses. Diante de dois tipos de inimigo e duas respostas possíveis, a chance de acerto por tentativa tendia a oscilar em torno de 50% quando não havia conhecimento prévio, tornando a evolução lenta e frágil a variações de sequência. Para mitigar esse efeito, foi adotado um esquema populacional com múltiplos indivíduos por geração. Cada indivíduo mantinha sua própria rede e seu próprio valor de *fitness*; após a eliminação, eram armazenados pesos, *fitness* e marca do cronômetro, possibilitando seleção e reprodução dos desempenhos mais promissores nas etapas evolutivas.

A formação de novas gerações preservava a diversidade por meio de mutações controladas, evitando homogeneização precoce e favorecendo a descoberta incremental de combinações de pesos mais eficazes.

Foram avaliadas funções de ativação usuais em redes alimentadas adiante, como Sigmoide, ReLU e Tangente Hiperbólica. Observou-se um desempenho discretamente superior com ReLU em camadas ocultas e na saída, razão pela qual essa configuração foi adotada nos experimentos principais.

Para efeito de contraste metodológico, registrou-se que Pereira (2023) empregou seis entradas, uma camada oculta com doze neurônios e duas saídas para pular e abaixar, treinando por Estratégias Evolutivas com avaliação baseada em tempo de sobrevivência. No presente estudo, a rede utilizou três entradas específicas do *runner* do dinossauro, duas camadas ocultas de seis neurônios cada e duas saídas, com avaliação orientada a ações corretas. A diferença de representação e de arquitetura buscou equilibrar simplicidade de sensores com capacidade de decisão, mantendo o foco no conjunto de inimigos terrestres e voadores.

4.7 Implementação da interação entre a RNA, o dinossauro e o cenário

O trabalho de Pereira (2023) utilizou Estratégias Evolutivas com avaliação centrada em tempo de sobrevivência, uma arquitetura com seis entradas, uma camada oculta com doze neurônios e duas saídas, priorizando mutações e seleção dos mais aptos. Nesta implementação, a avaliação foi orientada a ações corretas, e a rede possuía três entradas alinhadas ao cenário do *runner* e duas camadas ocultas de seis neurônios cada, e o processo evolutivo combinava seleção por roleta, recombinação e mutação. Essa configuração foi escolhida para investigar como operadores de seleção e recombinação, aliados a um *fit-*

ness comportamental, afetariam a aprendizagem de agentes em ambientes com inimigos terrestres e voadores.

O ambiente de execução foi projetado de forma dinâmica e estocástica. A cada rodada, uma população de 150 dinossauros era inserida em um mesmo cenário e exposta a sequências de obstáculos geradas procedural e aleatoriamente. Foram considerados dois tipos de inimigos, em conformidade com o comportamento do jogo original: inimigos terrestres e inimigos voadores. A resposta esperada por parte dos agentes era binária e relacionada ao eixo vertical: diante de um inimigo terrestre, o agente deveria realizar um salto; diante de um inimigo voador, a ação correta era abaixar-se.

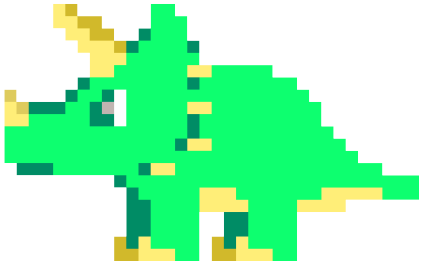

Figura 15 – Inimigo terrestre.

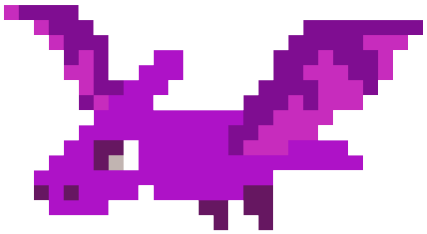

Figura 16 – Inimigo voador.

O acoplamento percepção–decisão–ação ocorria em *ticks* de aproximadamente 16 ms, o que correspondia à cerca de 60 quadros por segundo. Cada agente calculava, a cada quadro, as três entradas da rede neural: distância horizontal Δx, diferença de altura Δy e velocidade dos inimigos. Essas variáveis formavam um campo de visão simplificado e eram atualizadas enquanto o obstáculo permanecia na região útil de detecção. A rede produzia duas ativações de saída, associadas às ações *pular* e *abaixar*. Um limiar de decisão era aplicado para converter ativações em comandos, evitando ambiguidade de resposta.

O *feedback* era imediato e orientado a comportamento. Ações corretas no momento adequado incrementavam o *fitness* do indivíduo, enquanto que colisões promoviam a eliminação imediata. Ao ser eliminado, cada agente registrava pesos, *fitness* e marcava do cronômetro para uso do processo evolutivo.

Como os obstáculos se aproximavam a diferentes velocidades, a decisão correta dependia de sincronismo. O sistema considerava uma janela de acerto, que era a função de Δx e da velocidade do inimigo. As respostas fora dessa janela elevavam a chance de colisão, punindo atrasos e ações precipitadas. Esse desenho favorecia o aprendizado de tempo de reação e não apenas a classificação do tipo de obstáculo.

Em variantes descritas por Pereira (2023), diferentes mecânicas e sensores foram explorados. Neste, o recorte foi mantido propositalmente compacto para avaliar como a arquitetura com três entradas e duas camadas ocultas impactaria a tomada de decisão binária no eixo vertical.

Figura 17 – A primeira geração do AE.

O cenário foi controlado por um cronômetro interno que avançava a cada quadro. Esse contador organizava dois comportamentos principais: a geração de inimigos em intervalos regulares de quadros e a progressão de velocidade do jogo ao longo do tempo. O efeito combinado funcionava como um currículo de dificuldade. No início da rodada, a velocidade era menor e os intervalos eram mais previsíveis. Com o avanço do cronômetro, a velocidade aumentava e as janelas temporais de acerto se estreitavam, exigindo decisões mais precisas.

A detecção de colisão utilizava retângulos de *bounding box* para personagem e inimigos. As interseções entre áreas equivaliam ao impacto, acionando o mecanismo de eliminação. O mesmo sistema garantia o contato consistente com o chão e impedia a ultrapassagem das bordas da tela, de forma que a rede só precisasse aprender decisões no eixo vertical.

Ao término da rodada, os indivíduos eram ranqueados pelo *fitness* acumulado. A próxima geração era formada por recombinação dos mais aptos com introdução de mutações controladas, preservando elite para manter conhecimento já adquirido. Como consequência prática, as primeiras gerações tendiam a ter eliminação rápida. À medida que a população evoluia, observavam-se sequências mais longas de decisões corretas mesmo em velocidades elevadas.

A seguir, serão relatadas as principais diferenças de implementação do trabalho de referência em relação a este Trabalho de Conclusão de Curso.

4.8 Principais diferenças da implementação de Pereira (2023)

No experimento de Pereira, a rede neural gerenciou seis entradas, uma camada oculta com doze neurônios e duas saídas associadas às ações pular e abaixar. O treinamento utilizou Estratégias Evolutivas, com população inicial de aproximadamente dois mil indivíduos. O *fitness* foi definido pelo tempo de sobrevivência no cenário, isto é, quanto mais tempo o agente permanece ativo, maior a pontuação. Ao final de cada geração, os indivíduos eram ranqueados e o melhor era selecionado. A geração seguinte era composta por cópias do melhor (clonagem), seguidas de mutações aplicadas para manter diversidade e evitar estagnação. Os resultados reportados por Pereira indicaram melhora progressiva do comportamento ao longo das gerações.

Para fins de análise comparativa no capítulo de resultados, primeiramente foi implementada uma versão inspirada no estudo do Pereira. Essa versão reproduziu os princípios centrais do método de referência, incluindo *fitness* por tempo de sobrevivência, clonagem do melhor indivíduo e aplicação de mutações, sem a utilização do código original.

Além disto, este trabalho adotou técnicas diferentes para o AE desenvolvido, destacadas a seguir para evidenciar as diferenças entre as abordagens.

- Sensoriamento e arquitetura da rede: três entradas diretamente ligadas ao contexto do *runner* inimigo.getX() - playerIA.getX() inimigo.getY() - playerIA.getY() e velocidade dos inimigos; duas camadas ocultas com seis neurônios cada; duas saídas para pular e abaixar.
- Cenário e inimigos: apenas inimigos terrestres e voadores. Não havia movimentação lateral nem outros tipos de ameaça.
- Critério de *fitness*: pontuação orientada a comportamento, isto é, acúmulo por ações corretas realizadas no tempo adequado, em vez de medir exclusivamente o tempo de sobrevivência.
- Algoritmo Evolutivo: seleção por roleta proporcional ao *fitness*, recombinação para gerar descendentes, mutação com taxa controlada e elitismo para preservar a melhor fração da população.
- Parâmetros operacionais: experimentos com 30 gerações e população de 20 indivíduos, com avaliação *in-game*.

Em síntese, os dois trabalhos investigaram agentes autônomos no mesmo ambiente, porém com métricas, operadores evolutivos e arquiteturas distintas.

4.9 Considerações finais

O capítulo apresentou a construção integral do sistema em Java, desde a interface e a mecânica do jogo até os módulos de decisão. Foram definidos cenários e inimigos alinhados ao jogo original, restritos a ameaças terrestres e voadoras, com controle vertical simples e progressão de dificuldade guiada por um cronômetro. A arquitetura da rede neural foi especificada com três entradas diretamente relacionadas ao estado do ambiente, duas camadas ocultas com seis neurônios e duas saídas para pular e abaixar. O Algoritmo Evolutivo foi estruturado com seleção por roleta proporcional ao *fitness*, recombinação para geração de descendentes, mutação controlada e elitismo para preservação de soluções de alto desempenho.

Também ficaram claros os contrastes em relação à implementação de Pereira (2023): a diferenciação nos sensores de entrada, na profundidade da rede, no critério de *fitness* orientado a ações e nos operadores evolutivos utilizados. Com essa base técnica consolidada, o próximo capítulo apresentará os experimentos e analisará, de forma sistemática, o efeito dos parâmetros evolutivos e das escolhas arquiteturais sobre o desempenho dos agentes no jogo do dinossauro.

5 Resultados

Neste capítulo, serão relatados os resultados obtidos ao longo da etapa de experimentação do jogo do Dinossauro, desenvolvido neste Trabalho de Conclusão de Curso.

O conjunto de experimentos foi realizado em duas etapas: (i) primeiramente, foram executados testes para definir a arquitetura mais adequada da RNA para o problema em questão; (ii) em seguida, foram efetuados testes para avaliar o desempenho do Algoritmo Evolutivo implementado e selecionar parâmetros capazes de equilibrar a exploração do espaço de busca e a preservação de boas soluções. Nestes cenários, foi mantida a configuração da RNA referente à melhor arquitetura obtida nos primeiros experimentos: com três entradas, duas camadas ocultas de seis neurônios e duas saídas que correspondem às ações de pular e abaixar. Esta análise concentrou-se em dois fatores de ajuste do AE: a taxa de mutação e o tamanho da população. Para cada combinação de parâmetros foram realizadas dez execuções independentes, sempre com trinta gerações, registrando ao final de cada execução os valores do pior *fitness* e do melhor *fitness* da última geração.

Todos os experimentos foram executados em um desktop com processador Intel® Pentium® 5405U @ 2,30,GHz, 12,GB de memória RAM e Windows 10 (64 bits). A implementação foi realizada integralmente em Java, utilizando o Oracle OpenJDK 22.0.2 na IDE IntelliJ IDEA Community Edition 2024.2.1, apenas com bibliotecas padrão da linguagem.

Os códigos-fonte estão disponíveis em: <https://github.com/Guilherme0202PM/minhaBibliotecaJogosDinossauro>

5.1 Experimento com a Rede Neural Artificial

Nesta etapa, comparou-se o desempenho de duas arquiteturas de rede mantendo constantes todos os demais parâmetros do Algoritmo Evolutivo: taxa de mutação em 10%, taxa de elitismo em 20%, população de 150 indivíduos e seleção por roleta. As duas configurações para a RNA avaliadas foram com uma única camada oculta e duas camadas ocultas.

Tabela 1 – Resultados com uma RNA uma camada oculta com 6 neurônios.

Execução	**Pior *fitness***	**Melhor *fitness***
1	62	3877
2	25	1866
3	77	3837
4	96	4096
5	**94**	**4198**
6	62	4017
7	29	3995
8	11	1972
9	82	3936
10	134	3307
Média	**67,2**	**3456,1**

Na arquitetura com uma camada oculta (Tabela 1), o melhor *fitness* atingiu 4198 (Rodada 5). A média do melhor *fitness* nas 10 execuções foi 3456,1, enquanto a média do pior *fitness* foi 67,2. Observou-se ainda que apenas duas execuções encerraram com melhor *fitness* abaixo de 2000 (Rodadas 2 e 8).

Tabela 2 – Resultados com uma RNA com duas camadas ocultas com 6 neurônios.

Execução	**Pior *fitness***	**Melhor *fitness***
1	77	4138
2	114	3634
3	117	3672
4	136	3578
5	82	3859
6	77	3647
7	**111**	**4207**
8	7	1412
9	111	3600
10	77	3267
Média	**90,9**	**3501,4**

Na arquitetura com duas camadas ocultas (Tabela 2), verificou-se um ganho com melhor *fitness* de 4207 (Rodada 7). Neste caso, porém, a média do melhor *fitness* foi 3501,4. Apenas uma execução terminou com melhor *fitness* abaixo de 2000 (Rodada 8), caracterizando maior regularidade.

Portanto, as duas arquiteturas produziram picos muito próximos, mas a configuração com duas camadas ocultas (6 neurônios cada) apresentou média ligeiramente superior no melhor *fitness*, além de menos casos abaixo de 2000. Portanto, ela foi adotada como padrão nas análises subsequentes.

5.2 Experimento com a taxa de mutação

O objetivo neste experimento foi identificar um ponto de equilíbrio entre diversidade e estabilidade, variando as taxas de mutação.

Foram testados três valores em blocos de dez execuções: 3%, 5% e 10%. As Tabelas 3, 4 e 5 apresentam, para cada execução, os indicadores da última geração (pior e melhor *fitness*). Para estes experimentos, foram utilizados a taxa de elitismo de 20% e método de seleção por roleta.

Tabela 3 – Desempenho do AE com taxa de mutação de 3%.

Execução	**Pior *fitness***	**Melhor *fitness***
1	14	35
2	**115**	**811**
3	39	174
4	32	333
5	29	688
6	8	572
7	8	61
8	22	494
9	22	158
10	8	500
Média	**29,7**	**382,6**

A Tabela 3, com taxa de mutação de 3%, mostra a média do melhor *fitness* com valor de 382,6, enquanto o pior *fitness* ficou com uma média de 29,7. O teto (811) apareceu como caso isolado e a variabilidade relativa foi alta, indicando baixa previsibilidade do ganho.

Tabela 4 – Desempenho do AE com taxa de mutação de 5%.

Execução	**Pior *fitness***	**Melhor *fitness***
1	**92**	**1322**
2	39	96
3	8	178
4	137	612
5	8	1313
6	149	413
7	8	335
8	22	602
9	8	576
10	8	75
Média	**47,9**	**552,2**

A Tabela 4 evidencia os valores com a taxa de mutação de 5%, onde houve um avanço no teto e na média do melhor *fitness* (média de 552,2) em relação a 3%, mas a

dispersão permaneceu elevada com pior *fitness* com média de 47,9. Pode-se notar que esses resultados melhoraram em relação ao experimento anterior, mas ainda houveram valores baixos para o *fitness*.

Tabela 5 – Desempenho do AE com taxa de mutação de 10%.

Execução	**Pior *fitness***	**Melhor *fitness***
1	**137**	**2743**
2	127	486
3	39	552
4	8	546
5	22	71
6	103	2258
7	22	365
8	8	82
9	22	593
10	39	295
Média	**52,7**	**799,1**

Na Tabela 5, com taxa de mutação de 10%, são apresentados dois picos (2743 e 2258). Pode-se perceber que a média do melhor *fitness* sobiu para 799,1, comparado ao resultado anterior. O pior *fitness* teve média 52,7. Isto indicou que esta configuração teve capacidade de alcançar melhores valores de *fitness* em relação às taxas de mutação anteriormente testadas, com uma média também melhor, sendo a taxa escolhida para os próximos experimentos.

5.3 Experimento com a taxa populacional

Com a taxa de mutação em 10% e elitismo em 20%, esta etapa avaliou o efeito do tamanho da população por geração. Foram comparadas quatro configurações: 50, 100, 150 e 200 indivíduos, sempre em 30 gerações.

As Tabelas 6, 7, 8 e 9 apresentam, para cada execução, o pior *fitness* e o melhor *fitness* ao final da última geração.

Tabela 6 – Desempenho do AE com população de 50 indivíduos por geração.

Execução	**Pior *fitness***	**Melhor *fitness***
1	65	533
2	**63**	**3145**
3	39	1009
4	39	1150
5	8	910
6	95	1634
7	8	2580
8	8	1063
9	8	830
10	8	1167
Média	**34,1**	**1402,1**

Na Tabela 6, com o tamanho da população igual a 50, o melhor *fitness* apresenta média igual a 1402,1, com pico de 3145 (Rodada 2). O pior *fitness* teve média igual a 34,1.

Tabela 7 – Desempenho do AE com população de 100 indivíduos por geração.

Execução	**Pior *fitness***	**Melhor *fitness***
1	8	863
2	24	1364
3	152	3374
4	39	1027
5	**115**	**4093**
6	115	4033
7	73	2014
8	46	576
9	46	794
10	8	918
Média	**62,6**	**1905,6**

Na Tabela 7, com tamanho de população igual a 100, pode-se notar que o melhor *fitness* sobiu para média igual a 1905,6 e pico de 4093 (Rodada 5). O pior *fitness* teve média igual a 62,6. Observou-se, portanto, uma melhoria nos valores obtidos do teste anterior.

Tabela 8 – Desempenho do AE com população de 150 indivíduos por geração.

Execução	**Pior *fitness***	**Melhor *fitness***
1	77	4138
2	114	3634
3	117	3672
4	136	3578
5	82	3859
6	77	3647
7	**111**	**4207**
8	7	1412
9	111	3600
10	77	3267
Média	**90,9**	**3501,4**

Na Tabela 8, com população de tamanho de 150, pode-se observar que o melhor *fitness* atingiu média igual a 3501,4 com pico de 4207 (Rodada 7). O pior *fitness* teve média igual a 90,9. Apenas uma execução mostrou o melhor *fitness* abaixo de 2000 (1412), caracterizando uma maior robustez neste configuração.

Tabela 9 – Desempenho do AE com população de 200 indivíduos por geração.

Execução	**Pior *fitness***	**Melhor *fitness***
1	97	4055
2	**94**	**4258**
3	77	486
4	96	4137
5	153	3990
6	60	3776
7	60	4009
8	62	3586
9	79	3674
10	62	3761
Média	**84,0**	**3573,2**

Na Tabela 9, com tamanho de população igual a 200, aparece o maior pico absoluto (4258, Rodada 2) e a maior média do melhor *fitness* (3573,2). O pior *fitness* teve média igual a 84,0. Apesar do teto ligeiramente superior, a dispersão do melhor *fitness* foi maior que em 150, e houve uma execução claramente pior (com melhor igual a 486), sugerindo volatilidade. Por este motivo, 150 foi considerado o tamanho de população mais adequado para este problema diante dos experimentos realizados.

5.4 Comparação com a literatura

Este experimento teve como objetivo comparar o Algoritmo Evolutivo desenvolvido nesta pesquisa com a replicação do modelo proposto por Pereira (2023). A configuração adotada neste trabalho foi utilizando seleção por roleta, *crossover* e elitismo, enquanto a pesquisa de Pereira aplicou um esquema simplificado de clonagem do melhor. Neste esquema, apenas o indivíduo com maior *fitness* na geração era clonado para preencher a próxima população e, em seguida, os clones sofrem mutação, ou seja, o processo evolutivo dependia exclusivamente da clonagem do melhor e a mutação.

Mantendo a mutação de 10% e população de 150 indivíduos, a Tabela 10 resume os resultados do esquema de clonagem do melhor, enquanto a configuração adotada neste estudo pode ser observada na Tabela 8, com a melhor configuração obtida nos experimentos relatados anteriormente.

Tabela 10 – Desempenho do AE com clonagem do melhor indivíduo utilizada pelo Pereira (2023).

Execução	**Pior *fitness***	**Melhor *fitness***
1	530	953
2	162	937
3	530	953
4	530	567
5	240	290
6	530	573
7	170	229
8	149	208
9	**630**	**1244**
10	149	208
Média	**362**	**616,2**

Pode-se verificar que os valores dos melhores *fitness* no esquema de clonagem do melhor apresentou média igual a 616,2 para o melhor *fitness*, com pico de 1244 (Rodada 9) (Tabela 10). Por outro lado, a configuração deste trabalho obteve média igual a 3501,4, com pico de 4207 (Rodada 7) (Tabela 8). Em termos relativos, a média do melhor *fitness* foi cerca de 3,4 vezes maior, indicando maior robustez entre execuções.

Na clonagem do melhor, o pior *fitness* teve média igual a 362,0 (Tabela 10); enquanto que na configuração deste trabalho, a média do pior *fitness* foi 90,9 (Tabela 8). Isto pode ser um efeito da clonagem, em que replicava sempre o melhor indivíduo da população, ou seja, o melhor indivíduo direcionava a evolução em torno dele mesmo a cada geração. Por outro lado, como não havia o processo de seleção por roleta e recombinação, a diversidade gerada era penalizada a cada geração, reduzindo a variabilidade da população à medida que as gerações passavam.

Portanto, no contexto deste jogo, pode-se concluir que o esquema de clonagem do melhor, inspirado em Pereira (2023), gerou populações homogêneas com desempenho melhor entre os piores *fitness*, mas reduzido em relação aos melhores *fitness*; enquanto a configuração com roleta, *crossover* e elitismo produziu picos muito mais altos e média superior do melhor *fitness*. Para o objetivo deste estudo, que era maximizar os valores de aptidão das soluções, os dados favoreceram a configuração deste trabalho (Tabela 8) frente à clonagem do melhor (Tabela 10).

5.5 Considerações finais

Os primeiros experimentos referentes à arquitetura da RNA indicaram que, mantendo constantes os parâmetros do Algoritmo Evolutivo, a RNA com duas camadas ocultas de 6 neurônios apresentou desempenho melhor do que com uma camada, com média do melhor *fitness* de 3501,4 contra 3456,1, além de um piso mais alto para o pior *fitness* (média igual a 90,9 contra 67,2) (Tabela 1, Tabela 2).

Em relação aos parâmetros testados para o Algoritmo Evolutivo, a taxa de mutação de 10% apresentou o melhores resultados de média do melhor e do pior *fitness*. Quanto ao tamanho da população, o experimento com 150 indivíduos ofereceu o melhor equilíbrio entre pico e estabilidade: média do melhor *fitness* de 3501,4, muito próxima da média obtida com 200 indivíduos (3573,2), porém com menor dispersão e sem o *outlier* presente com população de 200 (Tabela 8, Tabela 9). As populações menores (50 e 100) ficaram aquém, com médias do melhor *fitness* de 1402,1 e 1905,6 , respectivamente (Tabela 6, Tabela 7).

Por fim, na comparação com o esquema de clonagem do melhor inspirado em Pereira (2023) (sem seleção por roleta, *crossover* nem elitismo), a configuração adotada neste estudo (seleção por roleta + *crossover* + elitismo) teve desempenho superior: média do melhor *fitness* de 3501,4 contra 616,2, além de pico de 4207 frente a 1244 (Tabela 8, Tabela 10). Ambos os modelos de AGs mostraram funcionalide no contexto do jogo, podendo-se perceber que a clonagem gerou populações mais homogêneas com média do pior *fitness* mais alta, enquanto a configuração proposta maximizou a capacidade de sobrevivência ao combinar preservação de bons indivíduos e mesclar características com a recombinação, garantindo a variabilidade genética útil.

Em síntese, a configuração recomendada para o cenário estudado foi: duas camadas ocultas de 6 neurônios, seleção por roleta com *crossover*, mutação de 10%, elitismo de 20% e população de 150 indivíduos por geração. Essa combinação equilibrou diversidade e preservação das soluções, garantindo o progresso evolutivo de geração a geração e promovendo resultados com médias elevadas e menor dispersão relativa.

6 Conclusão

Este Trabalho de Conclusão de Curso investigou heurísticas computacionais para o jogo do dinossauro do *Google Chrome* combinando Redes Neurais Artificiais e Algoritmo Evolutivo, sob protocolo experimental controlado.

O Algoritmo Evolutivo, no contexto do ambiente deste jogo, gerenciou uma população de indivíduos (dinossauros) associados a Redes Neurais Artificiais. O AE ajustava os pesos/vieses dessas redes a cada geração, a fim de maximizar o *fitness* de cada indivíduo da população. Neste trabalho, foram empregados os seguintes métodos: a seleção por roleta, o *crossover* por média elemento a elemento (vetor de pesos e vieses) e o elitismo com taxa de 20%. A taxa de mutação e o tamanho da população foram definidos empiricamente após uma sequência de experimentos, estabelecendo 10% como a taxa mais adequada para mutação e 150 como o tamanho indicado para população. Todos os experimentos relativos aos parâmetros do AE foram executados com o máximo de 30 gerações. Com esta configuração, o AE obteve um desempenho promissor, superando os resultados obtidos pela replicação de Pereira (2023) ao ser executado sob as mesmas condições que o primeiro, maximizando os valores de *fitness*.

Dessa forma, o Algoritmo Evolutivo (AE) foi capaz de treinar as Redes Neurais Artificiais (RNAs) ao longo das gerações. Paralelamente, essas RNAs atuaram como controladores em tempo real dos dinossauros. A cada quadro do jogo, as redes mapeavam o estado local do ambiente com base em três entradas: (i) distância horizontal até o obstáculo, (ii) diferença de altura entre o agente e o obstáculo, e (iii) velocidade global dos inimigos. As saídas da RNA correspondiam a duas possíveis ações: pular ou abaixar. Os pesos e vieses da rede permaneciam fixos durante toda a vida do indivíduo, enquanto a função de *fitness* era acumulada sempre que a ação executada coincidisse com a janela de acerto do obstáculo. Em resumo, as RNAs definiram a política de decisão de cada agente com base no estado atual do jogo. Segundo os experimentos realizados, os melhores resultados foram obtidos com a RNA contendo três entradas específicas do *runner* do dinossauro, duas camadas ocultas de seis neurônios cada e duas saídas, que correspondem às ações de pular e abaixar.

Além dos resultados experimentais comprovarem o sucesso das heurísticas computacionais para o problema em questão, este trabalho tem como contribuição para comunidade científica a disponibilização do código implementado: <https://github.com/Guilherme0202PM/minhaBibliotecaJogosDinossauro>. Como proposta para trabalhos futuros, pretende-se explorar outras arquiteturas de Redes Neurais Artificiais, incluindo o aumento do número de camadas ocultas e a avaliação de técnicas voltadas à mitigação do

sobreajuste. Além disso, busca-se investigar o uso de diferentes operadores genéticos, com ênfase na aplicação de variantes dos métodos de recombinação e na geração da população inicial — aspecto pouco abordado nesta pesquisa.

Referências

AL-GHARAIBEH, J. **Genetic Algorithms with Heuristic Knight's Tour Problem**. 2007. Online. Citado 2 vezes nas páginas 17 e 19.

BARRETO, J. M. **Introdução a Redes Neurais Artificiais**. [S.l.]: Local de publicação, 2002. Citado 2 vezes nas páginas 23 e 25.

BRAGA, A. d. P.; CARVALHO, A. P. L. F.; LUDERMIR, T. B. **Redes Neurais Artificiais: Teoria e Aplicações**. 2ª. ed.. ed. Rio de Janeiro: LTC - Livros Técnicos e Científicos, 2007. Citado 7 vezes nas páginas 8, 21, 22, 23, 24, 29 e 30.

COSTA, P. D. P.; PRAMPERO, P. S.; SALAZAR, Z. C. **Inteligência Artificial aplicada a Jogos Digitais**. 2009. Faculdade de Engenharia Elétrica e de Computação – FEEC, Universidade Estadual de Campinas – UNICAMP. Tópicos em Engenharia de Computação VI. Campinas, dezembro de 2009. Citado 3 vezes nas páginas 8, 29 e 30.

DARWIN, C. **On the Origin of Species**. London: John Murray, 1859. Citado 2 vezes nas páginas 10 e 14.

FACELI, K.; CASTRO, J. F.; MALABARBA, M. R. **Inteligência Artificial: Uma abordagem de Aprendizado de Máquina**. [S.l.]: Livros Técnicos e Científicos Editora LTDA, 2011. Citado na página 21.

GABRIEL, P. H. R.; DELBEM, A. C. B. **Fundamentos de algoritmos evolutivos**. São Carlos: ICMC-USP, 2008. <https://repositorio.usp.br/directbitstream/7472618b-87b3-4077-a1ca-eb5f40a0542c/nd_75.pdf>. Acesso em: 25 agosto. 2024. Citado 4 vezes nas páginas 10, 14, 16 e 17.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. [S.l.]: W. H. Freeman, 1979. ISBN 9780716710455. Citado na página 14.

GASPAR-CUNHA, A.; ANTUNES, C. H.; TAKAHASHI, R. **Manual de computação evolutiva e metaheurística**. Brasil: Editora UFMG, 2012. Citado 4 vezes nas páginas 14, 15, 18 e 20.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. [S.l.]: Addison-Wesley, 1989. Citado 2 vezes nas páginas 16 e 19.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. [S.l.]: The MIT Press, 1992. Citado 2 vezes nas páginas 16 e 18.

JONG, K. D. A history of evolutionary computation. In: FOGEL, D. B.; SCHWEFEL, H.-P. (Ed.). **Handbook of Evolutionary Computation**. Oxford: Oxford University Press, 2000. cap. 6, p. 40–58. Citado na página 15.

Laboratório iMobilis - UFOP. **Fundamentos de Redes Neurais**. s.d. <https://www2.decom.ufop.br/imobilis/fundamentos-de-redes-neurais/>. Acesso em: 14 outubro. 2024. Citado 2 vezes nas páginas 5 e 25.

LACHTERMACHER, G. **Pesquisa operacional na tomada de decisões**. Rio de Janeiro: Campus, 2001. Citado na página 13.

LIMA, I.; PINHEIRO, C. A. M.; SANTOS, F. A. O. **Inteligência Artificial**. 1. ed.. ed. Rio de Janeiro: Campus, 2014. Citado 7 vezes nas páginas 10, 22, 23, 24, 25, 26 e 27.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943. Citado na página 21.

NETO, A. O. L. Histologia do sistema nervoso: Diversidade celular e suas localizações. **Revista Científica Multidisciplinar Núcleo do Conhecimento**, v. 5, n. 8, p. 74–93, Novembro 2017. ISSN 2448-0959. Citado na página 20.

OLIVEIRA, G. P. de. **Métodos de Inteligência Artificial aplicados em jogos baseados em turnos**. 2018. Universidade Federal de Uberlândia – UFU. Uberlândia, Brasil. Citado 3 vezes nas páginas 8, 29 e 30.

PEARL, J. **Heuristics: Intelligent Search Strategies for Computer Problem Solving**. [S.l.]: Addison-Wesley, 1984. ISBN 9780201055948. Citado na página 14.

PEREIRA, J. V. D. **Avaliação da eficácia das estratégias evolutivas no treinamento de redes neurais perceptron aplicadas a jogos digitais**. Dissertação (Mestrado) — Universidade Federal Fluminense, Instituto de Ciência e Tecnologia, Rio das Ostras, 2023. Citado 24 vezes nas páginas 3, 4, 5, 6, 8, 9, 10, 11, 12, 17, 22, 24, 30, 31, 32, 33, 35, 42, 43, 45, 46, 53, 54 e 55.

PRECHELT, L. Early stopping—but when? In: ORR, G. B.; MüLLER, K. (Ed.). **Neural Networks: Tricks of the Trade**. Berlin: Springer, 1998. p. 55–69. ISBN 3-540-65311-2. Citado na página 28.

RAHMAN, W. **Inteligência artificial e aprendizado de máquina**. São Paulo: Editora Senac São Paulo, 2024. Citado 2 vezes nas páginas 10 e 25.

RECHENBERG, I. **Evolutionsstrategie - Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution**. [S.l.]: Frommann-Holzboog, 1973. Citado 2 vezes nas páginas 16 e 17.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. **Learning Internal Representations by Error Propagation**. [S.l.]: MIT Press, 1986. Citado 2 vezes nas páginas 23 e 26.

SADAVA, D.; HILLIS, D. M.; HELLER, H. C.; BERENBAUM, M. R. **Life: The Science of Biology**. 9th ed.. ed. Sunderland, MA: Sinauer Associates; W.H. Freeman, 2009. Citado na página 20.

SCHWEFEL, H.-P. **Evolutionsstrategie und numerishe optimierung**. Tese (Doutorado) — Technical University of Berlin, Berlin, Alemanha, 1975. Citado 2 vezes nas páginas 17 e 18.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 1, p. 1929–1958, 2014. Citado 2 vezes nas páginas 27 e 28.

TAHA, H. **Pesquisa operacional**. 8. ed. São Paulo: Prentice Hall, 2008. Citado na página 13.