UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

MARCO AURÉLIO LIMA DE SOUSA

APLICAÇÃO DE UM MODELO NEUROEVOLUTIVO PARA SIMULAR JOGADORES: UM ESTUDO DE CASO COM O JOGO *ULTIMATE GUITAR SHOW*

RUSSAS

2022

MARCO AURÉLIO LIMA DE SOUSA

APLICAÇÃO DE UM MODELO NEUROEVOLUTIVO PARA SIMULAR JOGADORES: UM ESTUDO DE CASO COM O JOGO *ULTIMATE GUITAR SHOW*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Bonfim Amaro Junior

RUSSAS

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S697a Sousa, Marco Aurélio Lima de.
Aplicação de um modelo Neuroevolutivo para simular jogadores : um estudo de caso com o jogo Ultimate Guitar Show / Marco Aurélio Lima de Sousa. – 2022.
60 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2022.
Orientação: Prof. Dr. Bonfim Amaro Junior.

1. Jogos. 2. Redes Neurais. 3. Algoritmos Genéticos. 4. Aprendizado por Reforço. 5. Neuroevolução. I. Título.

CDD 005.1

MARCO AURÉLIO LIMA DE SOUSA

APLICAÇÃO DE UM MODELO NEUROEVOLUTIVO PARA SIMULAR JOGADORES: UM ESTUDO DE CASO COM O JOGO *ULTIMATE GUITAR SHOW*

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Bonfim Amaro Junior (Orientador)
Universidade Federal do Ceará (UFC)

Profa. Ms. Tatiane Fernandes Figueiredo
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcio Costa Santos
Universidade Federal do Ceará (UFC)

À minha família por ter acreditado em mim desde o começo, aos amigos que sempre me ajudaram nessa jornada e ao meu orientador, Prof. Dr. Bonfim Amaro Junior, sem o qual eu não teria concluído este trabalho.

AGRADECIMENTOS

Em primeiro lugar a Deus, por ter me dado saúde e entusiasmo para que eu pudesse persistir na elaboração deste trabalho.

Ao Prof. Dr. Bonfim Amaro Junior, por toda sua atenção e ajuda que foram vitais para mim. Sou grato por sua dedicação e paciência ao me ajudar durante todo o processo.

A minha mãe, Maria Das Dores, ao meu pai, Albino Leite, e minhas irmãs, Michelle Lima e Mislene Lima, por todo o seu apoio e generosidade em me ajudar.

A Maria Ricarla pelo incentivo e pelas palavras de apoio durante a elaboração deste trabalho.

Aos demais membros da minha família e amigos, por sempre acreditarem em mim e me ajudarem direta e indiretamente.

Às pessoas com quem convivi ao longo desses anos de curso, que me incentivaram e que certamente tiveram impacto na minha formação acadêmica.

A Universidade Federal do Ceará, por tudo que eu aprendi durante o desenrolar do curso e que serviu de base para a execução do trabalho.

“Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer.”

(Alan Turing)

RESUMO

Este trabalho apresenta um modelo de aprendizado de máquina que, após treinamento especializado, obtém a capacitação necessária para jogar o jogo *Ultimate Guitar Show* (desenvolvido pelo próprio autor). Para isto, foi utilizado redes neurais combinados com algoritmos genéticos em uma abordagem neuroevolutiva. No contexto desta obra, os algoritmos genéticos são usados na definição dos pesos sinápticos durante o treinamento de uma rede neural profunda. A rede neural, por sua vez, retorna um valor binário que representa uma decisão no jogo. Cada cromossomo é testado individualmente e, através de um cálculo específico, seu valor de fitness (nível de aptidão em jogar) é definido e atribuído a ele. Foram efetuadas sete rodadas de treinamento até que se alcançou, na última delas, a melhor configuração genética, resultando na produção de diversos cromossomos extremamente especializados no jogo. Passadas todas as fases de treinamento, testou-se então o cromossomo gerado de melhor fitness em uma rede neural dentro do ambiente do jogo real. Tal teste, feito tanto com a fase usada no treinamento como outras fases que a rede neural não conhecia, comprovou a ocorrência de aprendizado, pois em todas estas fases diferentes a rede neural apresentou o mesmo nível de acurácia em jogar de forma autônoma.

Palavras-chave: Jogos. Redes Neurais. Algoritmos Genéticos. Aprendizado por Reforço. Neuroevolução.

ABSTRACT

This work presents a machine learning model that, after specialized training, obtains the necessary skills to play the Ultimate Guitar Show game (developed by the author). For this, neural networks were used combined with genetic algorithms in a neuroevolutionary approach. In the context of this work, genetic algorithms are used to define synaptic weights during the training of a deep neural network. The neural network, in turn, returns a binary value that represents a decision in the game. Each chromosome is tested individually and, through a specific calculation, its fitness value (level of aptitude in playing) is defined and attributed to it. Seven rounds of training were carried out until the best genetic configuration was reached in the last one, resulting in the production of several extremely specialized chromosomes in the game. After all the training phases, the chromosome generated with the best fitness was then tested in a neural network within the real game environment. Such a test, carried out both with the phase used in training and other phases that the neural network did not know about, proved the occurrence of learning, since in all these different phases the neural network presented the same level of accuracy in playing autonomously.

Keywords: Games. Neural Networks. Genetic Algorithm. Reinforcement Learning. neuro-evolution

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo genético
IA	Inteligência Artificial
IP	*Internet Protocol*
NC	*Quantidade de Notas Consecutivas atingidos pelo cromossomo*
OBX	*Order-Based Crossover*
P	*Porcentagem de pontos atingidos pelo cromossomo*
PMX	*Partially Mapped Crossover*
RPG	*Role-Playing Game*
T	*Tempo do jogo*
TCP	*Transmission Control Protocol*
UGS	*Ultimate Guitar Show*
UOBX	*Uniform Order Based Crossover*

SUMÁRIO

1 INTRODUÇÃO

O mercado de jogos eletrônicos cresce à cada ano, com empresas que faturam receitas na casa dos bilhões de dólares. Segundo uma matéria da revista Superinteressante (2020), o lucro total desse mercado em 2015 foi maior que a indústria cinematográfica e fonográfica juntas, culminando em um faturamento impressionante de US$ 91 bilhões de dólares no ano seguinte. Todo esse sucesso de mercado abriu um leque enorme de possíveis atuações na indústria dos jogos, sendo que uma dessas se destaca por estar presente na maioria dos títulos de sucesso: a Inteligência Artificial (OLIVEIRA, 2013).

No decorrer do desenvolvimento da indústria, os jogos precisam de inteligências artificiais capazes de tomar decisões suficientemente convincentes a ponto de parecerem com escolhas tomadas por pessoas. Inteligências artificiais que falham neste aspecto costumam apresentar comportamentos incoerentes e sem lógica do ponto de vista humano, impactando, assim, tanto a jogabilidade quanto a opinião do próprio jogador sobre o jogo (RIBEIRO *et al.*, 2009).

Neste contexto, o presente trabalho apresenta um jogo chamado *Ultimate Guitar Show (UGS)*, desenvolvido pelo próprio autor deste trabalho. Ele se baseia em um outro jogo, conhecido na literatura estrangeira como *Guitar Hero* (HERO, 2005). Nele, cinco colunas de peças coloridas “deslizam” na tela, de cima para baixo, a fim de simular as notas tocadas pela guitarra em uma música real. O objetivo é fazer o máximo de pontos acertando o máximo de notas. Quando se erra, o instrumento da música que o jogador está tocando durante a fase para, e ele deixa de ganhar pontos relativos a nota não tocada. Uma quantidade significativa de erros seguidos leva ao *game over*.

O *Ultimate Guitar Show* é o jogo específico referenciado no tema; ele receberá uma inteligência artificial capaz de jogá-lo por conta própria. Para tal, usaremos redes neurais e o mecanismo de aprendizado será fundamentado nos algoritmos genéticos, modelando assim uma estratégia neuroevolutiva adequada e eficaz para esta tarefa.

A seguir, a Seção 1.1 descreve o objetivo geral, os objetivos específicos (Seçao 1.2), a motivação (Seçao 1.3) e a divisão de conteúdo desse documento, na Seção 1.4.

1.1 Objetivo geral

O objetivo geral desse trabalho é demonstrar o poder das redes neurais na simulação de jogadores em jogos digitais, e para tal, produziremos um modelo que será útil para jogos com temática parecida com o *Ultimate Guitar Show*.

1.2 Objetivos Específicos

- Elaborar e treinar uma inteligência artificial capaz de jogar as fases do jogo obtendo o máximo de pontos no final.
- Fazer com que a inteligência artificial produzida seja competente o suficiente para jogar com um ser humano no mesmo nível ou até melhor que ele.
- Desenvolver um trabalho que exponha informações úteis para que desenvolvedores de jogos possam usá-las, de maneira integral ou em partes, nas inteligências artificiais dos seus próprios projetos.

1.3 Motivação

Pelo fato de o jogo em questão ter um caráter multiplayer, surgiu, inevitavelmente, a ideia de colocar uma Inteligência Artificial (IA) interna para que o jogador possa jogar "contra a máquina". Tendo em vista que nem sempre o jogador conta com um oponente humano disponível no momento da partida, uma inteligência artificial integrada ao jogo possibilitará a existência de fases multiplayer sempre que necessário.

1.4 Roteiro do trabalho

A fundamentação teórica é apresentada (seção 2). Em seguida, trabalhos relacionados à Jogos Digitais, Inteligência Artifical, Aprendizado Profundo, Redes Neurais e Algoritmos Genéticos são abordados, a fim de realizar um comparativo entre o que esses trabalhos e essa pesquisa abrangem (seção 3). Por fim, uma proposta de solução relacionada ao contexto desse trabalho é apresentada, assim como as etapas percorridas para a elaboração da mesma (seção 4).

2 FUNDAMENTAÇÃO TEÓRICA

As seções seguintes trarão uma visão geral sobre jogos digitais (seção 2.1), listando tipos e categorias, bem como abordagens de modelagem. Em seguida, abordaremos os fundamentos da Inteligência artificial e suas diversas vertentes e implementações, como as Redes Neurais Artificiais e Algoritmos genéticos.

2.1 Jogos digitais

Os jogos sempre fizerem parte do cotidiano humano, desde épocas remotas. Por exemplo, os antigos egípcios, há cerca de 5000 anos atrás, criaram um jogo de tabuleiro chamado Sanet que usavam como passatempo (BRASIL, 2022). Estes tipos de jogos analógicos ainda são comuns nos dias de hoje, como xadrez e damas, por exemplo. No entanto, os jogos digitais estão tomando progressivamente mais espaço por oferecerem jogos completamente novos e também versões digitais dos jogos analógicos já existentes.

Os jogos digitais modernos podem ser divididos em oito categorias (LUCCHESE; RIBEIRO, 2009):

- **Estratégia:** Neste tipo de jogo, o sucesso do jogador depende da sua capacidade de decisão e raciocínio.
- **Simuladores:** Jogos que se baseiam na imersão do jogador em um ambiente virtual que emula um ambiente físico complexo.
- **Aventura:** Jogos baseados em enigmas explícitos; o sucesso do jogador depende de habilidades psicomotoras.
- **Infantil:** Jogos projetado para crianças, no intuito de educar e divertir com uso de quebra-cabeças e estórias.
- **Passatempo:** São jogos conhecidos como “casuais”, que não apresentam enredo complexo. Ao invés disso, o jogo apresenta quebra-cabeças simples e de solução rápida.
- ***Role-Playing Game* (RPG):** Versões digitais de jogos de RPG de mesa.
- **Esporte:** São baseados em jogos competitivos reais.
- **Educacionais:** Jogos que, independentemente do grupo em que se enquadram, focam em critérios didáticos e pedagógicos na transmisão de conceitos.

Em relação a modelagem do jogo, pode-se usar a seguinte abordagem de 3 camadas lógicas (LUCCHESE; RIBEIRO, 2009):

- **Gameplay:** Camada que define como serão as interações do jogo, incluindo as regras. Em outras palavras, é o conceito geral do jogo.
- **Implementação:** Camada relacionada aos recursos e dispositivos físicos no qual o jogo é jogado. No caso dos jogos digitais, a implementação é o próprio software do jogo.
- **Infraestrutura:** Camada relacionada a infraestrutura que é necessária para jogar o jogo. Nos jogos digitais, fazem parte dessa infraestrutura o computador no qual o jogo executa e os servidores remotos para jogos online.

As categorias de jogos anteriormente citadas podem possuir implementações de inteligência artificial para aprimorar a sua *gameplay*. A seção 2.2 trará uma introdução sobre o que é uma inteligência artificial e quais os seus tipos e classificações.

2.2 Inteligencia artificial

Até os dias de hoje, uma definição universal e definitiva sobre o que é “inteligência” ainda não foi encontrada. Este termo tem significados amplos, mas pode-se definir uma inteligência pelas propriedades exibidas por ela, como a capacidade de lidar com novas situações e solucionar problemas (COPPIN, 2013).

No âmbito digital, poderíamos definir uma inteligência artificial como um sistema que age de uma maneira que um observador externo consideraria inteligente. Entretanto, apesar de correta, essa definição não abrange todo o universo de aplicação das inteligências artificiais. Uma definição mais precisa, citado por (COPPIN, 2013) é a seguinte: "Inteligência artificial envolve usar métodos baseados no comportamento inteligente de humanos e outros animais para solucionar problemas complexos".

Estas são apenas duas definições dentre inúmeras propostas ao longo do tempo. A obra (RUSSELL; NORVIG, 2013) listou as principais definições conhecidas e separou em 4 grupos principais:

- Pensando como um humano.
- Agindo como seres humanos.
- Pensando racionalmente.
- Agindo racionalmente.

Os grupos 1 e 2 contêm as definições que medem o sucesso da IA em termos de fidelidade ao desempenho humano, enquanto os grupos 3 e 4 medem o sucesso comparando-o com um modelo idealizado de inteligência chamado de “racionalidade”.

2.2.1 Pensando como um humano

As inteligências artificiais pensadas seguindo esta abordagem se baseiam no estudo de como o pensamento humano funciona. Para tal, é necessário penetrar nos componentes internos da mente humana. Temos basicamente três métodos para fazer isso:

- **Introspecção:** Analisamos nossos próprios pensamentos à medida que eles vão se desenvolvendo.
- **Experimentos:** Analisando e estudando outras pessoas.
- **Imagens cerebrais:** Analisando imagens da atividade cerebral de um indivíduo.

Neste âmbito usa-se de forma extensa conhecimentos de ciência cognitiva. Ela, por conseguinte, reúne modelos computacionais de IA e técnicas experimentais de psicologia a fim de construir teorias precisas sobre todo o processo e funcionamento da mente humana.

2.2.2 Agindo como seres humanos

Nesta abordagem, as inteligências artificiais são projetadas e testadas através do Teste de Turing, que consiste em um interrogatório que um humano faz a máquina usando texto. Para cada pergunta feita, a IA dará uma resposta que será analisada posteriormente. Se o avaliador não conseguir descobrir se a resposta dada é de uma máquina ou de uma pessoa, então a inteligência artificial passou no teste.

Para passar no Teste de Turing, o computador precisa ter pelo menos 4 capacidades:

- **Processamento de linguagem natural:** Para permitir a comunicação entre a pessoa e a máquina através de um idioma natural.
- **Representação do conhecimento:** Para armazenar tanto o que ela já sabe quanto as informações que ela recebe.
- **Raciocínio automatizado:** Para fazer uso das informações armazenadas a fim de responder as perguntas e chegar em conclusões novas.
- **Aprendizado de máquina:** Para o mecanismo de adaptação; detectar padrões e ir além dos limites deles.

O teste original de Turing propositalmente descartava a interação física direta entre o interrogador e a máquina. No entanto, uma abordagem chamada "Teste de Turing Total" inclui sinais de vídeo e interações diretas, como o interrogador repassar um objeto para a máquina. Neste contexto, então, temos duas capacidades a mais:

- **Visão computacional:** Para permitir a detecção de objetos.
- **Robótica:** Para movimentação da máquina e manipulação de objetos.

2.2.3 Pensando Racionalmente

As definições que fazem parte desse grupo têm como plano de fundo a Lógica Aristotélica como base na construção de Inteligências Artificiais. Os estudos do filósofo grego Aristóteles deram início a um campo de estudos chamado "Lógica", na tentativa de codificar um "Pensamento Correto", ou seja, processos de raciocínios que não podiam ser refutados. Para isso, Aristóteles usava silogismos, conhecidos também como inferência, para formar padrões estruturais de argumentos que sempre resultam em conclusões corretas se as premissas também fossem corretas. O exemplo mais clássico de silogismo é a seguinte frase: "Sócrates é homem; todos os homens são mortais; então, Sócrates é mortal" (SANTOS, 2020).

Os lógicos do século XIX desenvolveram uma notação específica e precisa para representar as coisas do mundo real e suas relações. A tradição Logicista da inteligência artificial visa criar programas que operam internamente usando essa notação lógica. O objetivo é que esses programas consigam resolver esses problemas lógicos e sempre devolvam soluções corretas, criando assim sistemas inteligentes. Esta abordagem, no entanto, enfrenta diversos obstáculos: não é fácil traduzir conhecimentos informais em uma linguagem 100% lógica, e mesmo quando isso é possível, nem sempre há recursos computacionais suficientes para processar os dados.

2.2.4 Agindo Racionalmente

Este grupo de definições se baseiam na ideia de agentes racionais. Um agente racional age para alcançar o melhor resultado possível ou o melhor resultado esperado, quando existe incerteza.

Esta abordagem, assim como a citada na subseção 2.2.3, utiliza a lógica de inferências para a tomada de decisões, mas não se limita apenas a ela. Decisões inteligentes não se baseiam apenas na lógica crua, pois a inferência correta não representa toda a racionalidade possível. Por exemplo, existem situações específicas em que não existe uma ação correta a se realizar no

momento, mas a decisão precisa ser tomada mesmo assim. Ações de reflexo, como se afastar abruptamente de uma chapa quente, são tomadas sem o uso do processamento de inferências lógicas.

Apesar de a racionalidade perfeita em um ambiente computacional ser inviável devido a demanda gigantesca de comportamento, esta abordagem se mostra superior pois não se baseia apenas na inferência lógica para a tomada de decisão. Além do mais, o padrão de racionalidade é matematicamente bem definido, diferentemente dos padrões de comportamentos humanos e, portanto, podem gerar modelos de agentes que comprovadamente irão atingir seu objetivo.

2.3 Aprendizado de máquina

Esta área é de extrema importância para o campo da inteligência artificial. Ela é responsável por dar a um sistema computacional a capacidade de aprender de forma autônoma. Com dados suficientes e um algoritmo eficiente, pode-se projetar um algoritmo que aprenda padrões e também algoritmos que consegue fazer previsões baseados nestes dados (LOPES; BRAGA, 2017).

Os problemas de *Machine Learning* podem ser divididos nas seguintes subáreas principais (SILVA, 2020):

- **Problemas de Classificação:** São problemas que, quando resolvidos pelo algoritmo produzem um resultado qualitativo, ou seja, a solução encontrada para o problema deve pertencer a uma categoria ou classe específica. O classificador implementado, baseado em entradas de dados já previamente classificadas, produzem um resultado indicando qual classe os dados de entrada se enquadram. Para exemplificar, um classificador que é capaz de determinar se um paciente está doente ou não-doente está resolvendo um problema de classificação, pois o resultado, como visto, é qualitativo e limitado a essas duas categorias.
- **Problemas de Regressão:** São problemas similares aos de classificação. A diferença básica entre eles é em relação ao tipo de solução que deve ser encontrada. Enquanto nos problemas de classificação buscamos saber a categoria de determinado conjunto de dados por intermédio de uma resposta qualitativa, nos problemas de regressão procuramos uma resposta quantitativa, de valor numérico. Exemplificando, um classificador que é capaz de determinar o valor do salário de um indivíduo baseado na idade e nos anos de escolaridade está resolvendo um problema de regressão, pois o resultado esperado é numérico.

- **Problemas de Agrupamento:** Conhecidos também como *clustering*, são problemas que envolvem agrupamentos de observações em grupos chamados de clusters. As observações armazenadas em um *cluster* são similares entre si e diferentes entre as armazenadas em outros clusters. Não há nenhum tipo de rotulação, como acontece nos problemas de classificação, logo não existe clusterização certa ou errada. Por exemplo, um classificador que é capaz de agrupar fotos similares de animais em clusters está resolvendo um problema de agrupamento, mesmo que não tenha conhecimento prévio de qual animal está sendo apresentado.

Os tipos de aprendizado de máquina são normalmente divididos em três categorias: aprendizagem supervisionada, não-supervisionada e por reforço (BRUNIALTI *et al.*, 2015). A figura 1 mostra onde cada tipo de aprendizagem é usado, bem como as suas subcategorias:

Figura 1 – Tipos de aprendizado de máquina

Fonte: Elaborado pelo autor (2022).

Na **Aprendizagem Supervisionada**, o algoritmo efetua o ajuste de parâmetros ou pesos de um modelo a partir do valor de erro, que é obtido através da comparação entre a resposta mostrada pelo algoritmo e a resposta correta esperada. Para tal, é necessário dados de treinamento pré-classificados para possibilitar esta comparação (COPPIN, 2013, 249).

A aprendizagem supervisionada é utilizada para resolução de problemas de classificação e regressão. Nos problemas de classificação, o algoritmo tentará prever os resultados em uma saída discreta, enquanto nos problemas de regressão, em uma saída contínua (BARROS, 2016).

Na **Aprendizagem Não-supervisionada**, os algoritmos aprendem sem a intervenção humana. Enquanto na aprendizagem supervisionada o algoritmo aprende por comparação, na não-supervisionada o algoritmo aprende simplesmente pela exposição direta aos dados. O próprio algoritmo, então, encontrará os padrões existentes nos dados e efetuará a classificação dos mesmos (COPPIN, 2013, 249).

A aprendizagem não-supervisionada é normalmente utilizada para resolução de problemas de agrupamento, pois ela permite abordar problemas onde há pouca ou nenhuma ideia de como os resultados devem aparentar. Pode ser utilizada na redução do número de dimensões de um conjunto de dados para isolar dados referentes apenas aos atributos mais úteis. Pode ser usado também na detecção de tendencias (BARROS, 2016).

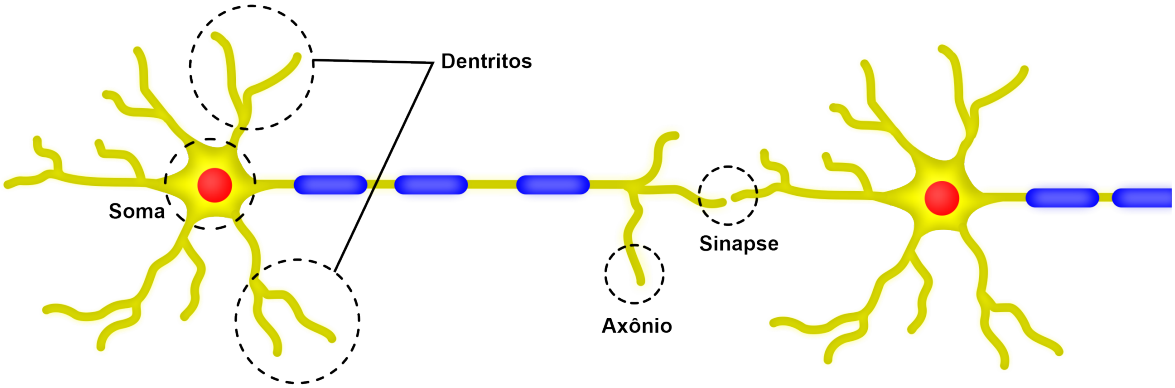
Já na **Aprendizagem por Reforço**, a aprendizagem ocorre por reforço positivo, se operar corretamente, e negativo, se operar incorretamente. O sistema de feedback avisa ao sistema de aprendizado que ele realizou a tarefa de forma correta ou incorreta, mas não diz exatamente o porquê e como ele acertou ou errou. Em posse da informação de feedback, cabe ao algoritmo a tarefa de atribuir responsabilidade do erro ou do acerto aos componentes e neurônios corretos (COPPIN, 2013, 250).

A aprendizagem por reforço é uma área de aprendizado de máquina que foi inspirada em psicologia comportamental. Também chamada de método de tentativa e erro, baseia a aprendizagem na experimentação do meio feita pelo agente. O agente, mesmo sem o conhecimento de quais ações são as melhores de serem executadas no momento, deve interagir com o ambiente para que melhore seu desempenho. Neste tipo de aprendizagem, o ambiente e as regras que o regem são desconhecidas pelo agente (MOTA, 2018).

2.4 Redes neurais

O sistema nervoso dos seres vivos multicelulares tem como objetivo coordenar as atividades dos diversos tecidos orgânicos, e para esse fim, usam tipos especiais de células chamadas neurônios ou células nervosas. Os neurônios tem a capacidade de reagir de forma imediata a estímulos externos e propagar este estímulo para outros neurônios através dos chamados impulsos nervosos. Tais impulsos viajam através das conexões que existem entre um neurônio e outro, chamado de sinapses. Estas sinapses, por sua vez, são formadas pelo encontro de um axônio e um dentrito.

Figura 2 – Esquema de um neurônio biológico



Fonte: Elaborado pelo autor (2022).

O neurônio artificial é análogo ao neurônio biológico, funcionando baseado nos mesmos princípios. É uma estrutura lógico-matemática que visa emular a forma, o comportamento e as funções da estrutura nervosa biológica. Comparativamente, podemos associar o dentrito à entrada de informação, a soma ao processamento e o axônio à saída.

Uma rede neural artificial é um agrupamento desses neurônios artificiais, formando uma rede com conexões fortes ou fracas entre eles, da mesma maneira que acontece no cérebro (FURTADO, 2019). Além do mais, como citado por (OLIVEIRA, 2021) há dois aspectos das redes neurais que os tornam similares ao cérebro humano:

- **O conhecimento é adquirido** por meio de um processo de aprendizagem, a partir do ambiente externo.
- **O conhecimento é armazenado** por meio da força de conexão entre os neurônios (pesos sinápticos).

De modo geral, as redes neurais podem ser divididas em três classes de arquitetura (OLIVEIRA, 2021):

- Alimentadas adiante de camada única.
- Redes alimentadas diretamente com múltiplas camadas.
- Redes recorrentes.

Em relação aos componentes fundamentais de uma rede neural básica, podemos listar as seguintes (LUCCHESE; RIBEIRO, 2009):

- **Conjunto de sinapses:** São as conexões entre os neurônios. Cada uma das conexões possuem um peso associado a ela.
- **Integrador:** Responsável por realizar as somas dos sinais de entrada, ponderados pelos pesos das conexões.

- **Função de ativação:** Restringe a amplitude e controla o valor de saída do neurônio.
- **Bias:** Valor associado ao neurônio usado para aumentar ou diminuir a entrada líquida da função de ativação.

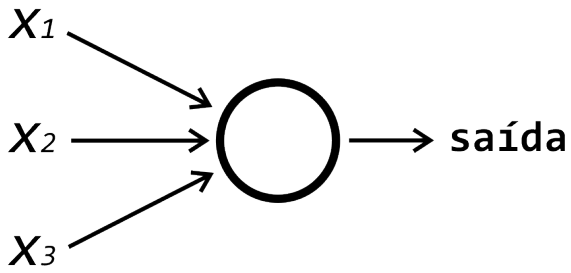
De acordo com o conceito supracitado, a aprendizagem de uma rede neural ocorre pelo ajuste de pesos sinápticos entre os neurônios. Para isso, é necessário um processo de estimulação pelo ambiente no qual ela está inserida. Esta estimulação, por sua vez, resulta em modificações nos parâmetros da rede, fazendo com que a mesma responda de maneira nova ao ambiente.

O efetivo aprendizado de uma rede neural ocorre quando esta produz uma solução suficientemente generalizada para um determinado problema. De forma geral, o treinamento de uma rede neural consiste em ajustar a matriz de pesos sinápticos a fim de que o vetor de saída apresente valores esperados para cada vetor de entrada (OLIVEIRA, 2021).

2.4.1 Perceptron

O perceptron, criado em 1958 por *Frank Rosenblatt*, consiste em um simples neurônio que classifica suas entradas em uma de duas categorias. Em outras palavras, as entradas de um perceptron podem ser de qualquer quantidade, mas sua saída apresenta apenas um valor binário (NIELSEN, 2019).

Figura 3 – Perceptron de *Frank Rosenblatt*



Fonte: Elaborado pelo autor (2022).

Normalmente o perceptron usa uma função degrau para definir a saída do neurônio (COPPIN, 2013). Se a soma ponderada da entrada $\boldsymbol{X}$ for maior que o valor do limiar $\boldsymbol{t}$, retornará **+1**, se for menor ou igual a $\boldsymbol{t}$, retornará **-1**:

$$X = \sum_{n=1}^{n} W_i X_i. \tag{2.1}$$

$$Degrau(X) = \begin{cases} +1 & \text{para x > t} \\ 0 & \text{para x <= t} \end{cases} . \tag{2.2}$$

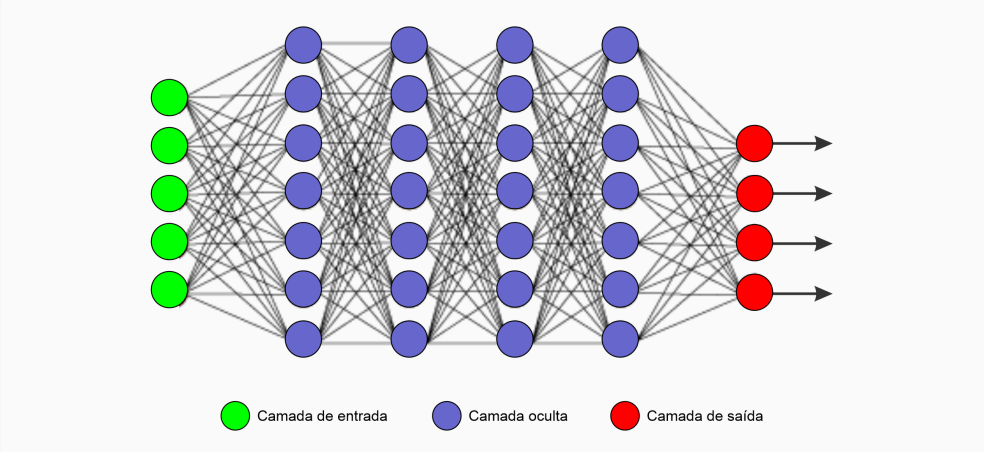
O treinamento de um perceptron é feito primeiramente pela atribuição de pesos aleatórios às entradas, normalmente valores entre **+0,5** e **–0,5**. Em seguida, um valor de treinamento é aplicado na entrada, e sua saída é observada. Se o valor exibido estiver incorreto, os pesos serão ajustados a fim de obter uma saída condizente com o valor esperado. Tal ajuste pode ser feito adicionando ou subtraindo valores.

2.4.2 Perceptrons de múltiplas camadas (MLP)

O perceptron de camada única só é capaz de classificar padrões que são linearmente separáveis. Esta é uma limitação que torna essa abordagem inapropriada para padrões mais complexos. Os perceptrons de múltiplas camadas, por outro lado, podem classificar padrões não-lineares (OLIVEIRA, 2021) e, portanto, estão entre os modelos neurais artificiais mais usados atualmente. Consistem em basicamente três camadas:

- **Camada de entrada:** Tem a tarefa de encaminhar as informações de entrada para as camadas ocultas da rede.
- **Camada oculta:** Separa a camada de entrada da camada de saída. Podem ser formadas por uma ou mais camadas, onde neurônios de uma mesma camada se comunicam diretamente com a camada posterior, mas não entre a mesma camada.
- **Camada de saída:** Camada onde a solução final é obtida.

Figura 4 – Arquitetura MLP com quatro camadas intermediárias



Fonte: Elaborado pelo autor (2022).

A arquitetura de múltiplas camadas é a base para o chamado *deep learning*, ou aprendizado profundo. Tal abordagem é extensivamente usada nos dias de hoje, em visão computacional e no reconhecimento e processamento da fala e linguagem natural (Figura 4).

2.5 Algorítmos genéticos

O Algoritmo genético (AG) é uma técnica de busca e otimização introduzida por *John Holland* em 1960, com a finalidade inicial de utilizar os conceitos de adaptação das espécies e seleção natural propostos por *Charles Darwin* na obtenção de melhorias na teoria computacional.

Na seleção natural, os indivíduos mais aptos e com maior longevidade tem maior probabilidade de se reproduzirem e transmitirem seus genes para as próximas gerações espécie. Os indivíduos que conseguem ter mais descendentes obtem, por conseguinte, mais chances de perpetuarem seus genes à frente por meio da sua prole. Esta abordagem natural garante que boas características sejam mantidas, enquanto as propriedades que são ruins para a espécie são descartadas (PACHECO, 2006).

Os princípios da seleção natural são a base da construção dos algoritmos genéticos, que analogamente a esta, buscam a melhor solução para determinado problema por intermédio da evolução de populações de soluções, que são codificadas por intermédio de cromossomos artificiais (PACHECO, 2006).

Muitas representações podem ser usadas no emprego de estratégias evolucionárias. Um cromossomo pode ser representado por uma cadeia de bits ou um vetor de valores de tamanhos iguais, onde cada bit ou valor é a representação de um gene. É considerada uma população um conjunto desses cromossomos, o qual é a representação completa de uma solução ou de uma classificação (COPPIN, 2013).

Tabela 1 – Relação entre a evolução natural e problemas computacionais

Evolução Natural	Problema Computacional
Indivíduo	Solção de um problema
População	Conjunto de soluções
Cromossomo	Representação de uma solução
Gene	Parte da representação de uma solução
Cruzamento	Operador de busca
Mutação	Operador de busca
Seleção natural	Reutilização de boas aproximações

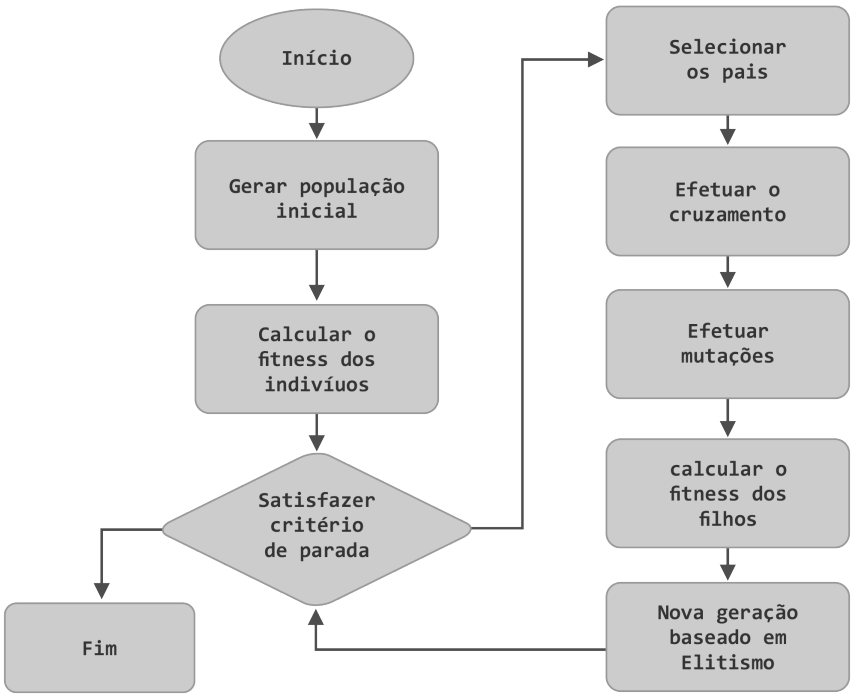
Fonte: Fonte: Elaborado pelo autor (2022).

A execução de um algoritmo genético basicamente é feita seguindo cinco passos, independente da representação que está sendo usada:

- Gerar uma primeira população, formada por cromossomos aleatórios.
- Determinar a aptidão de cada cromossomo.
- Se o critério de parada for satisfeito, parar. Se não, seguir para a próxima etapa.
- Aplicar cruzamentos e mutações nos cromossomos selecionados, a fim de definir uma nova população.
- Retornar a etapa citada no segundo ítem.

O seguinte fluxograma apresenta de forma visual e detalhada o fluxo de passos presentes no uso de um algorítmo genético:

Figura 5 – Fluxo detalhado de um AG



Fonte: Elaborado pelo autor (2022).

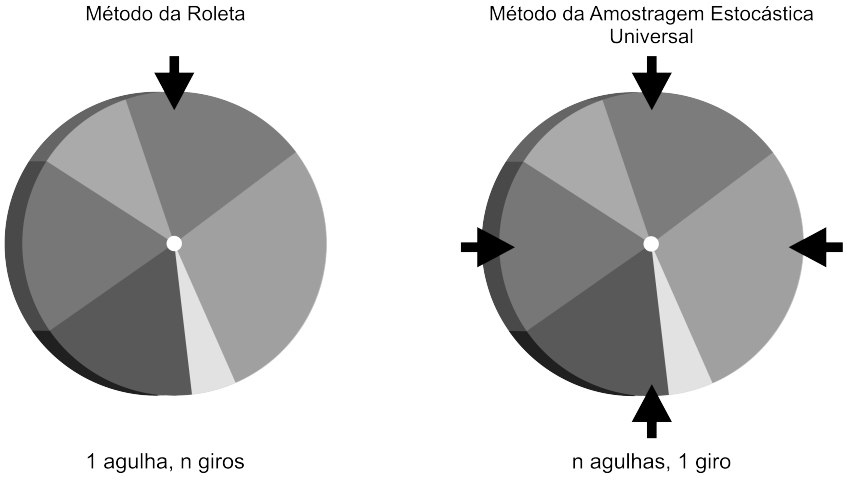
A **população inicial** é o início de todo o processo de evolução do algoritmo genético. Ela pode ser constituída por cromossomos aleatórios, onde cada um destes representa uma possível solução para o problema. Uma **etapa de avaliação** é necessária para filtrar as melhores soluções. Isto é feito por intermédio de uma função denominada **função de custo**, que avaliará o indivíduo quanto a sua aptidão em relação ao meio. Esta aptidão, por sua vez, é medida através do valor retornado pela função citada, chamado de ***fitness score***. A população, quando inteiramente avaliada, passa para a **etapa de seleção**, onde os indivíduos mais aptos permanecem

para a próxima geração enquanto os menos aptos são eliminados (TELES; GOMES, 2006).

Existem diversos tipos de seleção, sendo as mais utilizadas são (KATO, 2021):

- **Seleção por fitness:** Os indivíduos são selecionados levando-se em conta apenas o valor absoluto a ele atribuído pela função de *fitness*. Este tipo de seleção, entretanto, pode acabar resultando na chamada **convergência prematura**, onde os indivíduos mais aptos rapidamente ocupam toda a população, deixando o processo de busca focado em apenas uma região específica do espaço de busca. Tal fenômeno pode impedir o algoritmo genético de cobrir todas as possíveis soluções do problema.
- **Seleção por torneio:** Um número aleatório N de indivíduos é selecionado. Em seguida, um número aleatório R é gerado, sendo que seu valor varia entre 0 e 1. Definimos então um valor constante K, que servirá como um parâmetro de decisão (Por exemplo, 0.75). Se $R < K$, o indivíduo mais apto é selecionado, caso contrário o menos apto. Quanto maior o valor de K, maior será a pressão seletiva imposta na população.
- **Seleção por roleta:** Cada indivíduo da população é representado como um pedaço de uma roleta que será girada. Nesta abstração, o indivíduo ocupará um espaço na roleta proporcional a sua aptidão: quanto maior for, maior será a área reservada para ele e maior a probabilidade de ele ser selecionado. A roleta então é girada por um determinado número de vezes, e os indivíduos marcados em cada rodada serão os pais da próxima geração.
- **Amostragem Universal Estocástica:** Variação do método da roleta, utiliza n agulhas ao invés de apenas uma, sendo n o número de indivíduos que serão selecionados para a próxima geração. A roleta é então girada uma única vez.

Figura 6 – Métodos de seleção



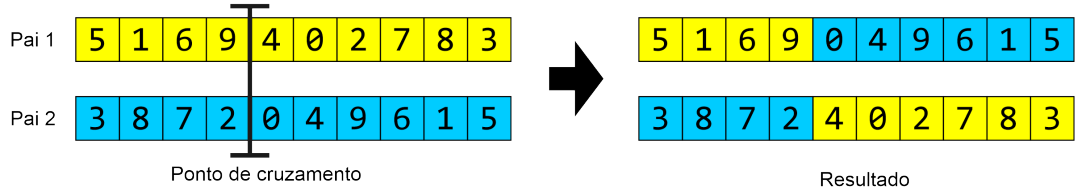
Fonte: Elaborado pelo autor (2022).

Após o processo de seleção, a próxima etapa é o de **cruzamento**, que serve para efetuar combinações genéticas entre um par de indivíduos na expectativa de que a geração resultante revele um ou mais indivíduos com um nível de aptidão maior do que seus pais (TELES; GOMES, 2006).

Os cruzamentos são feitos a partir da escolha de um ou mais pontos da cadeia cromossômica dos pais. Os valores separados por estes pontos formam blocos que serão combinados entre si para a geração de novos cromossomos (CARVALHO, 2009). O tipo cruzamento utilizado definirá a maneira que ocorrerá a mistura genética, e podem ser de basicamente 3 tipos (KATO, 2021):

- **Cruzamento de ponto simples:** seleciona dois indivíduos para o cruzamento e define um ponto único de separação, dividindo o cromossomo em 2 blocos. Em seguida, os blocos são combinados a fim de formar dois novos indivíduos:

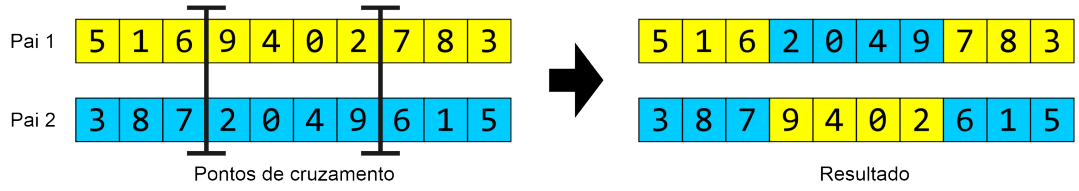
Figura 7 – Exemplo de cruzamento de ponto simples



Fonte: Elaborado pelo autor (2022).

- **Cruzamento multiponto:** Tem atuação similar ao de ponto simples, mas neste caso são criados mais de um ponto de cruzamento:

Figura 8 – Exemplo de cruzamento multiponto

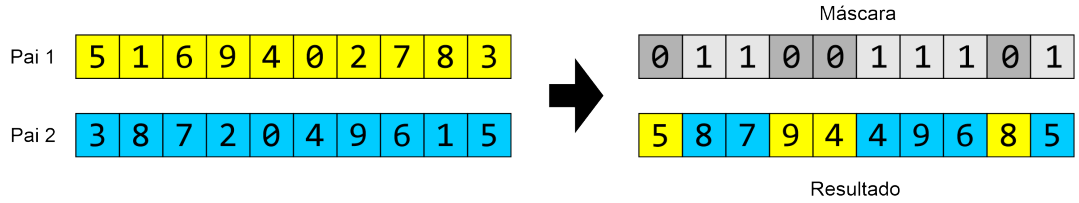


Fonte: Elaborado pelo autor (2022).

- **Cruzamento uniforme:** Esta abordagem acontece a nível de gene e não ao nível de segmento, como as anteriores. Primeiramente é definido qual o nível de participação cromossômica dos pais no cruzamento. Isto é feito definindo uma porcentagem de genes que serão usados por cada um deles na reprodução. Por exemplo, pode-se definir que serão usados no cruzamento 25% dos genes do pai 1 e 75% dos genes do pai 2. Em seguida é

gerado um vetor de bits aleatórios com o mesmo comprimento dos cromossomos dos pais, que funcionará como máscara. A quantidade de valores de 0 e 1 que aparecerão seguirá a proporção de participação já definida. No exemplo acima, por exemplo, seria gerado um vetor de bits ordenado aleatoriamente, com 25% de zeros e 75% de uns. Ocorrerá então a combinação de genes, onde bits 0 representam os bits do pai 1 e os bits 1 representam os bits do pai 2:

Figura 9 – Exemplo de cruzamento uniforme



Fonte: Elaborado pelo autor (2022).

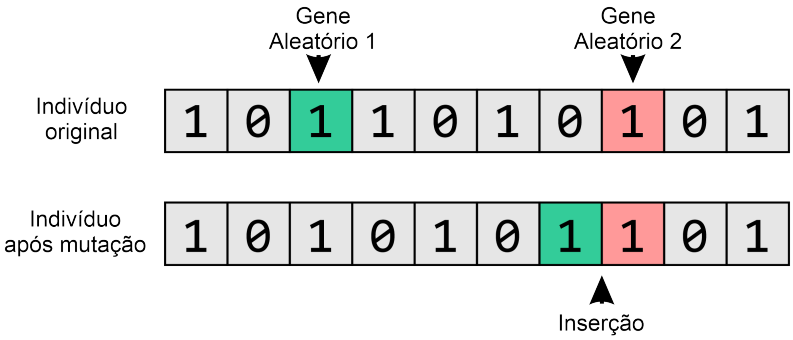
Para recombinações de cromossomos ordenados, onde uma toca direta não pode ser executada podemos usar alguns métodos, sendo três dos principais o ***Partially Mapped Crossover*** **(PMX)**, ***Order-Based Crossover*** **(OBX)** e ***Uniform Order Based Crossover*** **(UOBX)**. Estes métodos são especialmente usados quando o cromossomo é uma lista ordenada, como o caso de uma lista de cidades a ser percorrida no Problema do Cacheiro Viajante (LARRAÑAGA *et al.*, 1999).

Adicionalmente, podemos selecionar um grupo pequeno de indivíduos provenientes da etapa de cruzamento e efetuar pequenas **mutações genéticas**. Estas mutações são feitas através de alterações diretas na sequência de valores do cromossomo. Isto garante maior variabilidade genética e previne que o algoritmo acabe estagnado em um mínimo local. Contudo, essas mutações não podem ocorrer em uma frequência muito alta, pois isto resultaria em uma busca essencialmente aleatória. Por isso, a taxa de mutações que ocorrerão obedece a um valor probabilístico pré-definido, que normalmente varia de 0,001 a 0,01 (COPPIN, 2013).

As mutações podem ser de vários tipos, sendo as principais (KATO, 2021):

- **Mutação de inserção:** Este método seleciona dois genes aleatórios do cromossomo, e em seguida move o primeiro gene a fim de que este fique imediatamente atrás do segundo. Muito utilizada em problemas de permutação, não causa muita modificação na sequência do indivíduo.

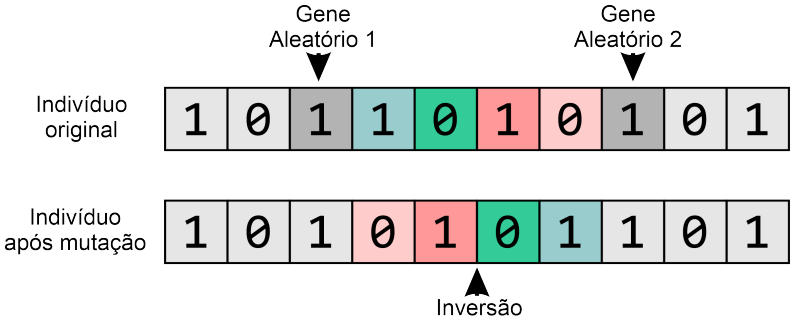
Figura 10 – Exemplo de mutação de inserção



Fonte: Elaborado pelo autor (2022).

- **Mutação de inversão:** são selecionados dois genes aleatórios, e então o segmento interno formado entre eles é invertido. Também é comumente utilizado em problemas de permutação.

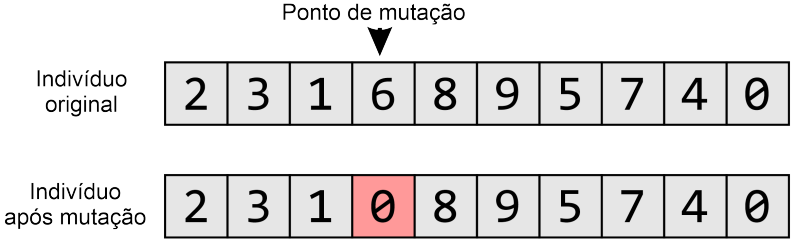
Figura 11 – Exemplo de mutação de inversão



Fonte: Elaborado pelo autor (2022).

- **Mutação uniforme:** Método amplamente utilizados para cromossomos que usam representação real e inteira na definição dos seus genes. Nele, o valor de um gene aleatório é mudado para um valor V também aleatório, sendo V um elemento do conjunto de valores que o gene pode assumir.

Figura 12 – Exemplo de mutação uniforme



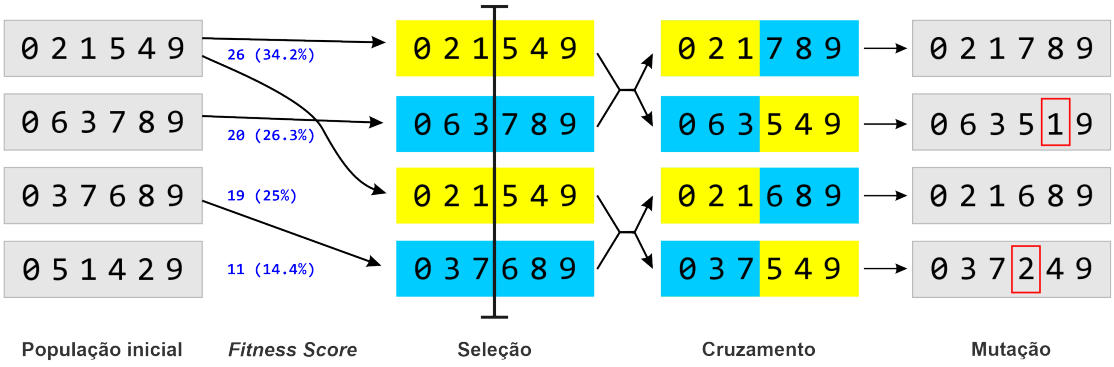
Fonte: Elaborado pelo autor (2022).

O **critério de parada** é utilizado para terminar a execução de um algoritmo genético em um determinado ponto de execução. Quando é possível identificar um padrão ótimo em relação aos indivíduos de certa geração, não é mais necessário continuar a execução do código. Por este motivo, este é um dos critérios de parada mais básicos. No entanto, pode acontecer de não ser possível definir se há um padrão ótimo nas populações geradas. Quando isso acontece, os seguintes eventos podem ser usados como critérios de parada:

- Tempo máximo de execução ou número de gerações excedeu um valor pré-definido.
- Número total de avaliações da função de fitness alcança um valor pré-definido.
- Os operadores genéticos já não estão mais produzindo melhorias significativas nas gerações subsequentes.

A figura 13 mostra as etapas na aplicação de um algoritmo genético na resolução de um problema. Em um território de 10 cidades, queremos ir da cidade 0 até a cidade 9 e, durante a viagem queremos visitar apenas 4 cidades ao todo. Estas 4 cidades devem fazer parte da rota que possui a menor distância entre a cidade de saída e a cidade de chegada. O algoritmo genético produzirá gerações de soluções possíveis até encontrar o cromossomo mais apto de todos, que representará a melhor solução para o problema.

Figura 13 – Processo de seleção, cruzamento e mutação em um AG no exemplo das cidades visitadas.



Fonte: Elaborado pelo autor (2022).

3 TRABALHOS RELACIONADOS

Nesta seçao serão abordados alguns trabalhos relacionados a inteligência artificial e aprendizado de máquina aplicados aos jogos digitais.

O trabalho de (SOUZA; VAHLDICKR, 2013) aborda como a indústria de games influenciou o progresso do campo da inteligência artificial. Tal influência foi decisiva neste processo, gerando duas abordagens diferentes: A IA acadêmica, que se esforça em tentar recriar a inteligência humana, e a IA para Jogos, que focam primariamente em melhorar a jogabilidade dos *games*. A natureza abstrata e convidativa dos jogos faz com que sejam um ambiente ideal para a aplicação de técnicas de Aprendizado de Máquina. Um feito marcante nesta área aconteceu no ano de 1997, onde uma inteligência artificial *Deep Blue* conseguiu derrotar o então melhor jogador de xadrez do mundo, *Garry Kasparov*. A demanda crescente por IAs nessa indústria fomentou pesquisas e desenvolvimento de técnicas inovadoras. Na tabela abaixo, temos uma lista de nove jogos e suas respectivas técnicas de IA:

Tabela 2 – Jogos lançados e suas repectivas técnicas de IA

Ano	Jogo	Técnica de IA
1952	OXO	Busca em árvore
1958	Jogo xadrez	Poda alpha-beta
1974	Qwak	Padrão de movimentação
1980	Pacman	Movimentação com personalidade
1989	SimCity	A-life (DNA virtual) e Autômata Celular
1990	Herzog Zwei	Máquina de estados finita, pathfinding
1996	Battlecruiser 3000AD	Redes neurais
2000	The Sims	Máquina de estados Fuzzy e A-life
2001	Black & White	Redes neurais e aprendizado por esforço

Fonte: Adaptado de (SOUZA; VAHLDICKR, 2013).

O trabalho (MEDINA; MüLLER, 2015) mostra o uso de um algoritmo genético para simular um jogador em um jogo de Damas. A abordagem utilizada nele usa um conjunto de etapas bastante usual e comum a todas as implementações de AG, sendo a principal diferença o fato de não usarem cruzamentos entre populações. Ao invés de reprodução, os autores deste trabalho optaram pelo uso extensivo de mutações para gerá-las. Outro ponto importante é que os indivíduos gerados não são aleatórios; eles são obtidos através de uma função de varredura de espaço de busca que retorna jogadas possíveis e válidas. Cada uma dessas possíveis jogadas representa um cromossomo, que é guardada em uma estrutura de armazenamento. Os genes que integram a estrutura dos cromossomos são formados por situações de jogadas, como número da pedra, número da pedra capturada e pontos da jogada. A AG usada neste trabalho não usa uma

função de fitness, pois os indivíduos gerados já tem um índice de adaptação, que se baseia na análise de quatro critérios, que se relacionam com a própria consequência da jogada: captura de pedra do oponente, perda ou não de pedra e jogada resulta em dama.

O trabalho (STELTTER, 2016) demonstra a utilização de um algoritmo puramente genético capaz de jogar o jogo Super Mario de forma autônoma. O algoritmo visa encontrar uma sequência de ações que possibilite o término das fases sem perder "vidas". O número de ações desta sequência não é conhecido a priori. Os testes executados foram baseados em 4 parâmetros: população inicial, quantidade de pais selecionados, quantidade de indivíduos criados pelos pais e quantidade de indivíduos mutantes. A condição de parada podia ser ativada em dois momentos: quando o objetivo fosse alcançado ou quando fossem passadas 1000 gerações. No teste 1 foram atribuídos os valores 100, 5, 20, 2, respectivamente. Contudo, tal configuração convergiu e o algoritmo começou a produzir filhos idênticos aos pais, uma vez que ambos os pais começaram a ter o mesmo cromossomo. Logo, o primeiro teste não foi eficiente e o AG não conseguiu finalizar a fase do jogo, sendo então necessário gradativamente modificar os parâmetros para valores maiores. Tal aperfeiçoamento resultou em sucesso na quarta configuração, que possuía os valores 600, 70, 600 e 100, respectivamente, com 133 gerações e cerca de 33 minutos de execução do algoritmo. O teste 5, por sua vez, usou valores de configuração idênticos ao último teste bem-sucedido, mas sem limitar o número de gerações durante 24 horas de execução. Os gráficos dos resultados deste teste mostraram que o algoritmo continuou evoluindo até a geração 350, sendo que a partir desse ponto a evolução foi mínima. Verificou-se, portanto, que quanto maior a quantidade de indivíduos presentes no algoritmo, maior é a evolução da população. No entanto, uma população maior também significa um maior tempo de execução.

O trabalho (MORAES, 2019) usa algoritmos genéticos e aprendizado por reforço para otimizar o desempenho de uma rede neural usada para controlar um jogador no jogo Pong. Os pesos da rede neural utilizada são os genes do algoritmo evolucionário. Foi empregada com entrada da rede neural as duas métricas referentes ao jogo, a saber: posição da barra do jogador e posição da bola. Não foram usados gradiente descendente, apenas cruzamentos e mutações de genes. A rede neural foi utilizada como agente pelo modelo de aprendizado por reforço, o qual teve como entrada 6 variáveis do jogo: posição x e y da bola, posição x e y da barra e posição x e y da bola no último estado. Os pesos sinápticos da camada oculta foram atualizados pelo algoritmo genético. A função de *fitness* recompensa os agentes com 1 ponto para rebote de bola, e 5 pontos para vitória contra o oponente. A recompensa vai sendo somada

até que o primeiro jogador alcance 20 pontos. Foram escolhidos arbitrariamente o número de 16 indivíduos por geração, onde a primeira é formada por redes neurais inicializadas com pesos aleatórios. As gerações subsequentes são formadas por cruzamentos e mutações dos indivíduos da geração anterior, sendo escolhidos 3 deles usando Elitismo. Cruzando esses 3 indivíduos, inserindo mutações e criando indivíduos aleatórios para cruzamentos, foi possível criar todos os 16 indivíduos requeridos para a geração. Os experimentos feitos usando essa estratégia se mostraram eficazes na produção de um modelo capaz de vencer qualquer adversário humano neste jogo. O algoritmo genético foi capaz de aprender a rebater a bola e fazer pontos em menos de 20 horas de jogo.

A obra (ALVES; ANDRADE, 2020) apresenta um modelo de algoritmo evolutivo capaz de controlar de forma autônoma uma nave no jogo Asteriods. Neste jogo, o jogador controla uma nave espacial de maneira multidirecional, em um campo repleto de asteroides e discos voadores. O objetivo é atirar nos asteroides e discos voadores a fim de destruí-los. A nave do jogador não pode tocar os asteroides, e não pode ser atingido por fogo inimigo proveniente dos discos voadores. A dificuldade do jogo aumenta à medida que o número de asteroides aumenta. Os autores posicionaram 10 sensores na nave do jogador a fim de detectar a presença de asteroides e discos voadores até uma determinada distância. Estes sensores operam de forma binária, detectando ou não a presença. Em seguida, uma combinação sensorial foi utilizada para identificar o índice do cromossomo baseado em quais sensores foram ativados. Em outras palavras, o vetor de 10 posições que armazena o estado dos sensores é mapeado a fim de produzir um valor inteiro que representa um índice específico do cromossomo que armazena ações a serem tomadas pela nave como atirar, rotacionar e acelerar. A função de avaliação avalia a solução através da quantidade de asteroides destruídas pela nave: quanto maior essa quantidade maior pontuação esta solução terá. O método de seleção usado foi por Torneio, e a recombinação dos cromossomos foi feito através de Cruzamento Uniforme. Ao final do último experimento, após cerca de 16 horas de jogo, o algoritmo apresentou resultados satisfatórios em relação a proposta do projeto.

4 METODOLOGIA

Nesta seção serão descritos os procedimentos e métodos planejados para a execução deste trabalho. Na Seção 4.1 abordaremos o funcionamento geral do jogo *Ultimate Guitar Show*, bem como suas características e regras. Em seguida, na seção 4.2 será explicado o funcionamento do software **Editor de Fases** do jogo. Também será explicado o que é o **Editor de Sequência** e como ele funciona. Em 4.3 será explicado como será a coleta de dados de treinamento da IA a partir do jogo, bem como a topologia da rede neural e a representação do cromossomo.

4.1 Funcionamento do *Ultimate Guitar Show*

O jogo *Ultimate Guitar Show* têm regras claras e bem definidas. O principal objetivo é simular uma guitarra ou qualquer outro instrumento contido na música, a fim de que o jogador tenha a sensação que o está tocando. Tal sensação se deve ao fato de que, se o jogador erra notas, seu instrumento para de tocar naquele exato momento, sobrando apenas o restante dos instrumentos do plano de fundo da fase. Em resumo, enquanto se acerta a sequência da fase no tempo correto, o instrumento tocará. Se houver algum erro por parte do jogador, o instrumento ficará mudo até que o mesmo comece a acertar novamente. A soma de 10 erros consecutivos leva ao *game over*.

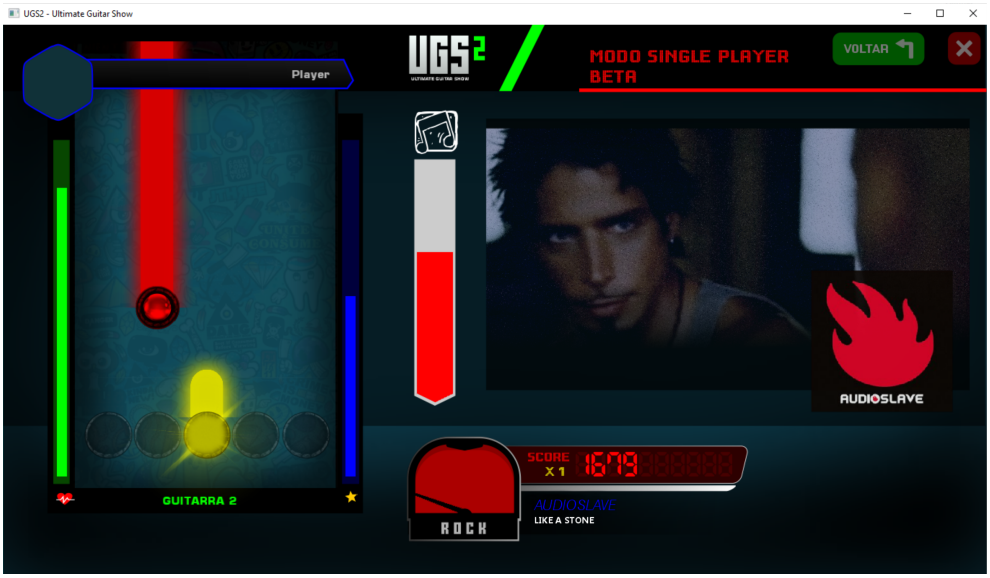
As consequências de um erro na fase do jogo não se limitam apenas ao fator auditivo. Tais erros também irão impactar na pontuação do jogador. O *Ultimate Guitar Show* apresenta duas métricas para representar o *score*:

- **Pontuação:** Cada nota acertada e cada *sustain* executado trará pontos para o jogador. Quanto mais precisa for a jogada, mais pontos valerá.
- **Notas consecutivas:** Representa a quantidade de notas acertadas consecutivamente sem erros entre elas (não implementada visualmente, mas acessível via código).

Cada nota presente na música é representada por um pequeno círculo colorido que desliza de cima para baixo. Cada cor representa uma tecla diferente que deve ser pressionada:

- **Verde:** Relativo à tecla **A** do teclado.
- **Vermelho:** Relativo à tecla **S** do teclado.
- **Amarelo:** Relativo à tecla **D** ou **J** do teclado.
- **Azul:** Relativo à tecla **K** do teclado.
- **Laranja:** Relativo à tecla **L** do teclado.

Figura 14 – Screenshot de uma fase do jogo Ultimate Guitar Show



Fonte: Elaborado pelo autor (2022).

O jogador acerta a nota quando consegue pressionar a tecla correta no exato momento em que ela passa por cima base estática (ver Figura 17). O alinhamento da nota com esta base coincide diretamente com o tempo correto que uma determinada nota musical real aparece na música. Algumas notas podem apresentar um rastro colorido, chamado de *sustain*, que representa o tempo que a nota dura. No jogo, a tecla só deve ser solta no momento em que a trilha de *sustain* acaba.

Para ser possível jogar com um instrumento específico de determinada música, é necessário separar as faixas de audio de cada um dos instrumentos em arquivos independentes. Para isto, foi utilizado um serviço online chamado Moises (SYSTEMS, 2021), que possibilita a separação da maioria dos instrumentos presentes nas músicas em várias faixas de audio. O Moises é um projeto brasileiro que usa IA para efetuar esta separação, e oferece o download do resultado em vários formatos de áudio diferentes. Para a preparação das fases do *Ultimate Guitar Show*, as faixas foram baixadas no formato **.m4a**, e então convertidas para **.ogg**, que é o formato de audio usado em todo o jogo. Esta conversão de formatos foi feita usando o software open source *Audacity* (AUDACITY, 2022).

4.2 Funcionamento do editor de fases

O editor de fases é um software projetado para produzir uma fase do jogo. Cada uma dessas fases é armazenada em uma pasta independente no local onde o jogo se encontra instalado. Tal pasta segue um padrão de organização estrutural pré-estabelecido, o que significa

que os arquivos e subpastas devem ter determinados nomes e estarem em lugares específicos para que o jogo reconheça como uma fase válida. A tarefa do editor é criar esta pasta com a estrutura exata exigida pelo jogo.

Para entender o funcionamento do editor, citaremos alguns conceitos e abstrações presentes na implementação do jogo:

- **Notas:** As notas são representadas no jogo por peças que surgem de cima para baixo em uma esteira. Há 5 colunas em que as peças podem surgir. Cada coluna gera peças de uma cor diferente, respectivamente: verde, vermelho, amarelo, azul e laranja. Nos arquivos de sequência, cada cor é representada por um valor inteiro, começando de 0 (verde) até o 4 (laranja).
- **Acordes:** Os acordes são representados no jogo como um conjunto de duas ou mais notas tocadas simultaneamente. O valor mínimo de um acorde é 2 e o máximo é 4.
- ***Sustain:*** Os *sustains* são notas ou acordes que devem ser tocados de maneira contínua pelo jogador. É representado no jogo por uma barra em cima da nota, que indica até que ponto a nota deve ser tocada.
- **Instrumentos:** O jogo oferece opções de até 15 instrumentos musicais diferentes, os quais são representados por um identificador numérico inteiro não-negativo. Qualquer música que use os seguintes itens pode ser utilizada como uma fase do jogo válida:

Tabela 3 – Instrumentos musicais e seus respectivos identificadores.

Código	Instrumento musical
0	Guitarra 1
1	Guitarra 2
2	Baixo
3	Bateria
4	Sanfona
5	Zabumba
6	Triangulo
7	Saxofone
8	Violão
9	Cavaquinho
10	Ukulele
11	Trompete
12	Sampler
13	Piano
14	Teclado

Fonte: Elaborado pelo autor (2022).

As músicas utilizadas pelo jogo se encontram dentro de **/songs**, e estão contidas nas subpastas lá presentes. Estas subpastas, por sua vez, são nomeadas com números inteiros, e

à medida que são adicionadas mais músicas, os nomes das pastas seguintes são o incremento do último valor inteiro utilizado. Não pode haver falhas neste sequenciamento numérico de nomeação de pastas.Se houver, todas as músicas depois da falha serão ignoradas. Por exemplo, se a nomeação estiver **/0 - /1 - /2 - /3 - /5 - /6**, o jogo só fará a leitura das fases até o valor 3, pois houve uma quebra da sequência. Cada música oferecerá até no máximo 4 instrumentos diferentes para o jogador tocar.

As pastas citadas acima segue a seguinte estrutura interna:

- **Song.ogg** - Música inteira sem modificações no formato **.ogg**. Vai ser usada no menu como preview quando o mouse ficar sobre o ícone da música.
- **info/about.txt** - Arquivo com as informações da fase. Deve seguir exatamente o seguinte formato, linha por linha:
  - **Nome do artista.**
  - **Nome da música.**
  - **Tempo total da faixa.**
  - **Dificuldade disponível.**, representado por uma sequência binária de tamanho 3. Cada bit representa uma dificuldade que está disponível respectivamente, sendo o valor 1 o indicador presença e o 0 indicador ausência. Por exemplo, para uma fase que tem apenas a opção de "Fácil"disponível, esta linha do arquivo irá conter a sequência “100”. Se contém todas as 3 dificuldades, será “111”.
- **Info/instrument(1,2,3,4).txt**
  - **‘Cod_instrumento’ + ‘NOME_DO_INSTRUMENTO’**
- **/audio** - Pasta que contêm as faixas de áudio usadas na fase do jogo. Consiste em um conjunto de arquivos com os áudios dos instrumentos previamente separados. O nome do arquivo de um instrumento deve seguir a tabela de identificadores mostrada anteriormente. Deve conter, obrigatoriamente, um arquivo de áudio chamado “background.ogg”, que á a faixa da música sem os instrumentos contidos nos demais arquivos. Por exemplo, uma música que tenha disponibilidade dos instrumentos guitarra 1, Baixo e Bateria apresentará os seguintes arquivos na pasta:
  - **0.ogg**
  - **2.ogg**
  - **3.ogg**
  - **background.ogg**

- **/picture** - Pasta que contém arquivos de imagem **.png** do artista ou banda. São 3 arquivos ao todo, com os seguintes nomes:
  - **card.ogg (148x148 px)** - Imagem da logo da banda usada nos cards do menu principal.
  - **logo.ogg (246x246 px)** - Imagem da logo usada no menu de escolha de instrumento e dificuldade.
  - **poster (1368x769 px)** - Imagem do artista ou banda usada no background do menu de escolha de instrumento e dificuldade.
- **/sequence** - Contém os ítens relativos as sequências de notas da fase. Dentro desta pasta estão as subpastas que contém as sequencias adaptadas aos níveis de dificuldade disponíveis. Se a fase não contém determinada dificuldade, ela pode ser simplesmente omitida sem prejuízo para o jogo. Segue o padrão:
  - **/0** - Pasta que contém as sequencias do tipo **fácil** para todos os instrumentos da fase.
  - **/1** - Pasta que contém as sequencias do tipo **médio** para todos os instrumentos da fase.
  - **/2** - Pasta que contém as sequencias do tipo **difícil** para todos os instrumentos da fase.
  - **speeds.txt** - Arquivo que contém a velocidade definida para cada dificuldade da fase. O arquivo contem uma sequência de 3 inteiros na mesma linha. Estes inteiros podem ser valores entre 6 e 11, onde o 6 representa a velocidade mais baixa e o 11 a mais alta.

Dentro das pastas de sequência citadas acima há os arquivos com a ordem de descida das peças do jogo. Cada instrumento contido na fase tem um arquivo separado com sua própria sequência. Os nomes dos arquivos de sequência devem seguir o código de instrumentos, e são no formato **.txt**. Por exemplo, para uma fase fácil que tenha instrumentos Guitarra 1, Baixo e Bateria, o padrão ficaria assim:

- **/sequence**
  - **/0.txt**
  - **/2.txt**
  - **/3.txt**

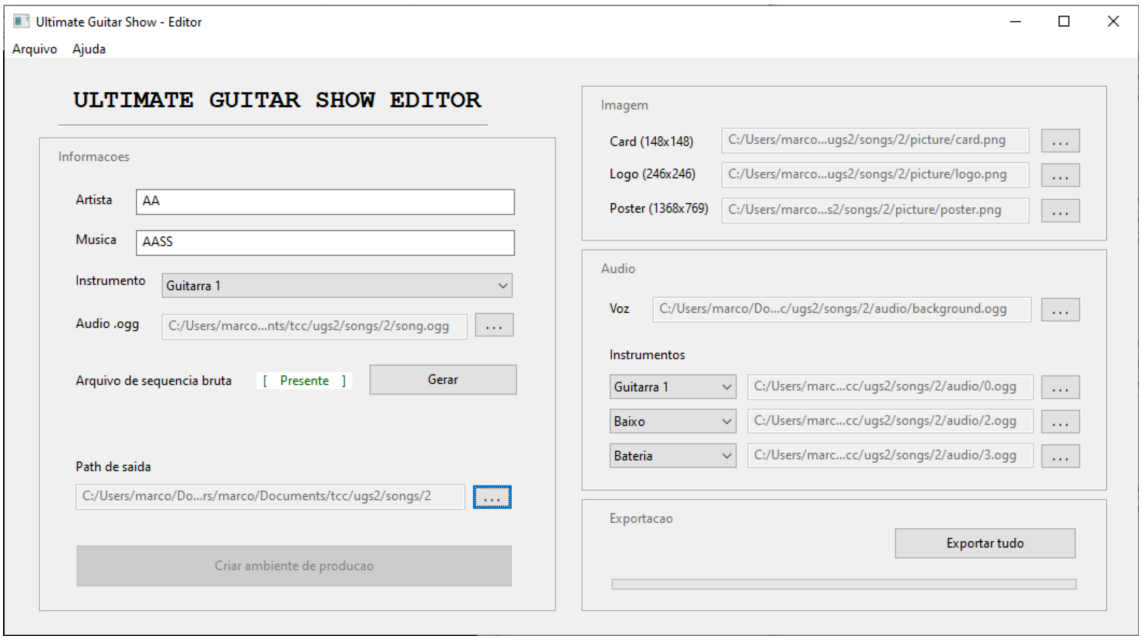
São estes arquivos de sequência que serão lidos diretamente pelo jogo. Ao se escolher a música, o instrumento e a dificuldade, o jogo se encarrega de encontrar o arquivo de sequência correto e inicia a fase.

O arquivo de sequência lido pelo jogo é um arquivo de texto comum onde cada linha representa uma nota ser produzida em tempo real pelo jogo. Cada uma dessas linhas é formada por tres valores separadas por espaço, que representam, respectivamente, o tempo em que a nota aparecerá na tela (tipo float), o identificador das notas (tipo int) e o valor de sustain (tipo float).

Há 3 dificuldades no jogo: fácil, médio e difícil. Elas têm as seguintes características:

- **Fácil:**
  - Usa apenas as 3 primeiras colunas do jogo.
  - Acordes podem ser nos tamanhos 2.
- **Médio:**
  - Usa apenas as 4 primeiras colunas do jogo.
  - Acordes podem ser nos tamanhos 2, 3 e 4.
- **Difícil:**
  - Usa todas as 5 colunas do jogo..
  - Acordes podem ser nos tamanhos 2, 3 e 4.

Figura 15 – Software editor de fases do jogo



Fonte: Elaborado pelo autor (2022).

O editor de fases (Figura 15) possui internamente o Editor de Sequência, que tem a finalidade de definir o local onde as notas serão tocadas, bem como se houve *sustain* de

notas ou acordes. Por este motivo, cada um dos instrumentos contidos na fase deve ser editado separadamente. A pessoa que manejará o Editor de Sequência deve conhecer bem a música para produzir uma boa edição.

O Editor de Sequencia é capaz de gerar a versão **Fácil**, **Médio** e **Difícil** de um determinado instrumento à partir de uma mesma edição. Para que isso seja possível, ao invés de gerar a sequência final no formato do jogo, será criado um arquivo intermediário chamado de **sequência bruta**. Este arquivo, depois de devidamente construído, poderá ser convertido para os formatos apropriados. A partir da sequencia bruta, é possível produzir uma fase em qualquer uma das dificuldades (fácil, médio e difícil).

O arquivo de sequência bruta gerado pelo Editor de Sequencia é um arquivo de texto comum onde cada linha representa uma nota. Cada uma dessas linhas é formada por tres valores separadas por espaço, que representam, respectivamente, o tempo em que a nota aparecerá na tela (tipo float), peso das notas (tipo int) e o valor de sustain (tipo float).

Vale ressaltar, como visto acima, que o arquivo de sequência bruta não guarda informações referentes a cor das notas e nem o tamanho dos acordes. Tal definição será feita na etapa de conversão de arquivo de sequência bruta. Esta etapa é responsável por produzir arquivos de sequência apropriadas para todas as dificuldades de um determinado instrumento a partir do arquivo de sequência bruta. A conversão ocorre respeitando as seguintes regras:

- A cor das notas é gerada aleatoriamente pela função de conversão; esta cor, como dito antes, é representada por um inteiro entre 0 e 4.
- Considerando que o peso das notas presentes no arquivo podem variar entre valores de 1 a 4, aplica-se o seguinte protocolo para a substituição dos pesos por notas e acordes válidos:
  - **Fácil**
    - **Peso 1:** Produzir uma nota de cor aleatória.
    - **Peso 2:** Produzir um acorde de 2 notas de cor aleatória.
    - **Peso 3:** Produzir um acorde de 3 notas de cor aleatória.
    - **Peso 4:** Produzir um acorde de 3 notas de cor aleatória.
  - **Médio**
    - **Peso 1:** Produzir uma nota de cor aleatória.
    - **Peso 2:** Produzir um acorde de 2 notas de cor aleatória.
    - **Peso 3:** Produzir um acorde de 3 notas de cor aleatória.
    - **Peso 4:** Produzir um acorde de 4 notas de cor aleatória.

- **Difícil**
  - **Peso 1:** Produzir uma nota de cor aleatória.
  - **Peso 2:** Produzir um acorde de 2 notas de cor aleatória.
  - **Peso 3:** Produzir um acorde de 3 notas de cor aleatória.
  - **Peso 4:** Produzir um acorde de 4 notas de cor aleatória.

Dentro da pasta de instalação do editor de fases há uma subpasta chamada /exports que tem a função de guardar os arquivos resultantes da conversão, bem como todos os arquivos de imagem e audio necesssárias pra a fase. Estes arquivos, depois de prontos, serão copiados para a pasta de instalação como uma fase válida do jogo.

Figura 16 – Editor de sequencia de notas

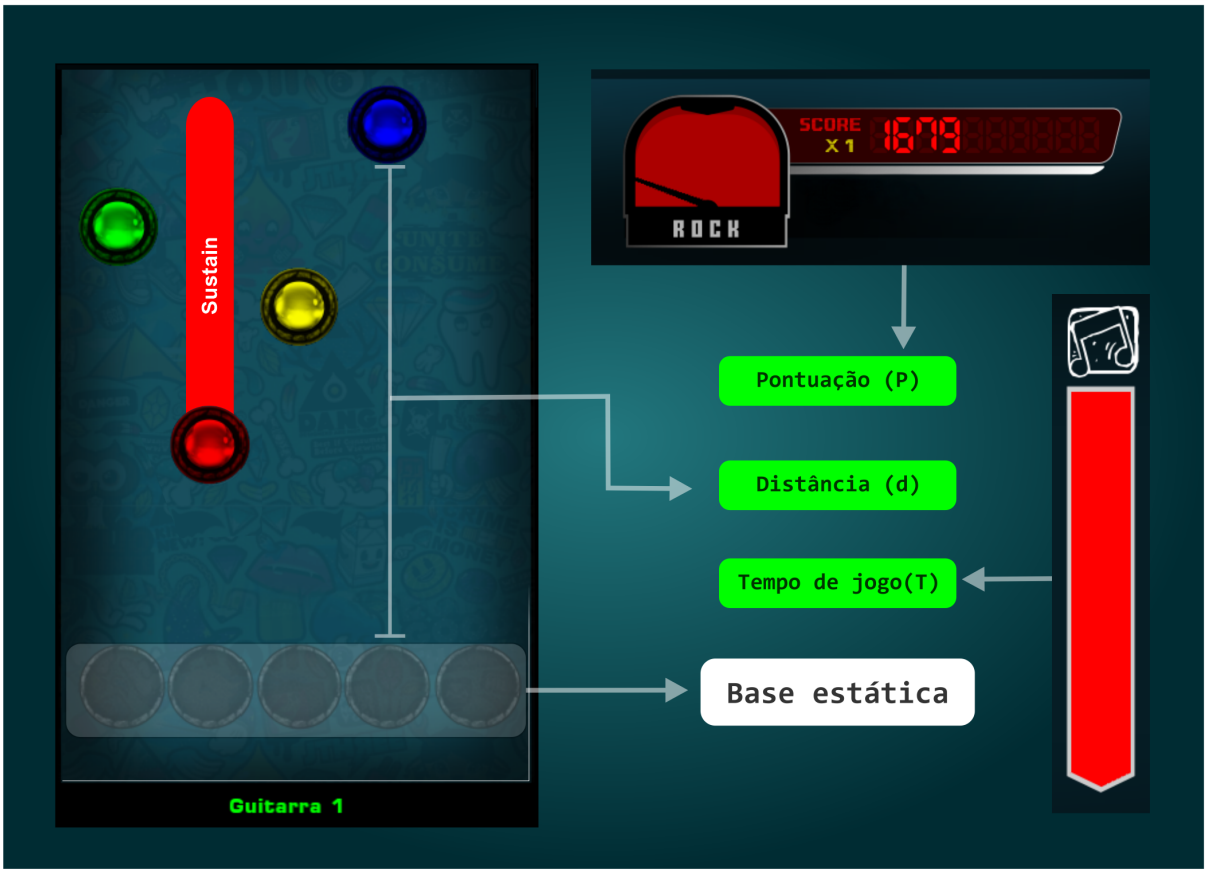
Fonte: Elaborado pelo autor (2022).

4.3 Coleta de dados e modelo neuro-evolutivo

O aprendizado ocorrerá no decorrer de várias gerações de treinamento a partir da evolução de cada individuo no algoritmo genético, princípio do modelo neuroevolutivo. Os valores dos pesos das conexões da rede neural constituirão os cromossomos. Os resultados serão apresentados por meio da definição dos parâmetros básicos do AG e do critério de aprendizado que vai ser fundamentado nos parâmetros suscitados a seguir.

Durante o treinamento, a rede neural receberá três informações do ambiente do jogo: a pontuação total (P), a distância em pixels da peça em relação a base estática (d) e o tempo de jogo (T). A **camada de entrada** será composta por 3 neurônios, que receberão estes três valores durante o jogo, respectivamente.

Figura 17 – Parâmetros de alimentação dos neurônios de entrada



Fonte: Elaborado pelo autor (2022).

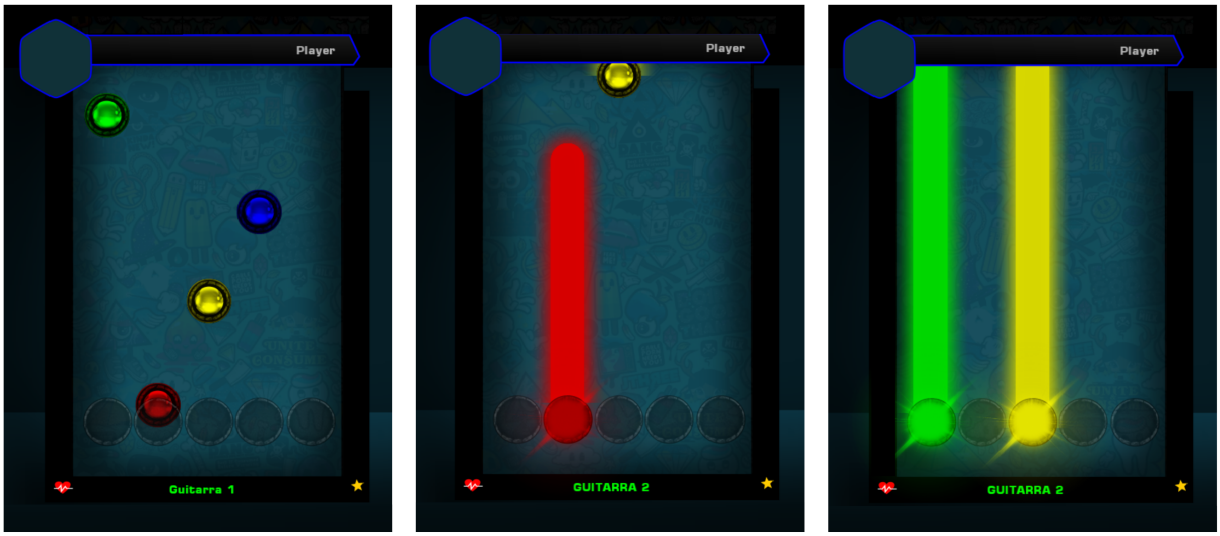
A eficácia da rede neural será medida, num primeiro momento, pela capacidade de jogar sem errar as notas. Em um segundo momento, ela será medida pela pontuação obtida em um dado trecho em comparação com um jogador humano.

A IA terá que ser capaz de decidir qual o melhor tipo de pressionamento de botão. Isto porque o jogo oferece três formas distintas de peças que representam as notas musicais, e consequentemente, três formas diferentes de pressionamento:

- **Nota simples:** Usado para representar uma nota musical. Para acertar, o jogador deve pressionar apenas um botão e logo em seguida soltar. Em notas simples, o valor de *sustain* é zero.

- **Nota simples com *sustain*:** Usado para representar uma nota musical com duração maior. Para acertar, o jogador deve pressionar apenas um botão e segurar até que a trilha colorida acabe.
- **Nota composta com *sustain*:** Usado para representar um acorde da música. Para acertar, o jogador deve pressionar vários botões ao mesmo tempo e manter esse pressionamento até que as trilhas coloridas acabem. As notas compostas podem ser formadas por até 3 notas simples.

Figura 18 – Tipos de notas presentes no jogo



Nota simples | Nota simples com *sustain* | Nota composta com *sustain*

Fonte: Elaborado pelo autor (2022).

A **camada oculta** da RN será formada por duas camadas, construídas com 3 neurônios cada. A definição da quantidade segue a seguinte fórmula:

$$HL = IL + \frac{IL}{2} \tag{4.1}$$

Onde **IL** representa o tamanho da camada de entrada, e **HL** o tamanho de cada camada oculta.

A **camada de saída** terá apenas 1 neurônio, o qual oferecerá um valor de tipo *float* que representa o pressionamento ou não de botões:

$$f(x) = \begin{cases} pressionar & \text{se } x >= 0.8 \\ soltar & \text{se } x < 0.8 \end{cases} . \tag{4.2}$$

Quando a IA decidir pressionar um botão em um determinado momento (x >= 0.8), tal pressionamento ficará ativo até que ela decida soltar o botão (x < 0.8). Em outras palavras, ela funcionará de forma idêntica a um interruptor simples, onde cada ação de ligamento necessita de uma ação explícita de desligamento. Esta abordagem possibilitará que a IA consiga acertar notas simples e compostas com a mesma facilidade, por usar uma abordagem uniforme para ambas.

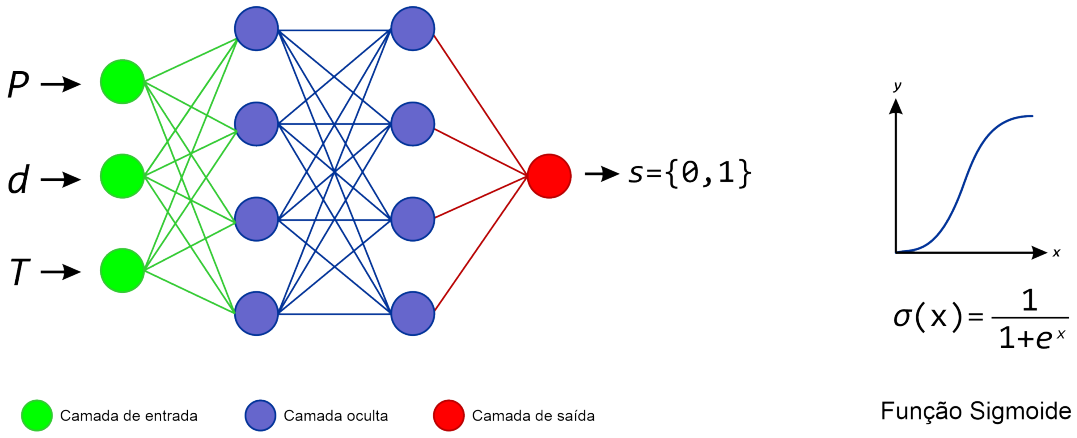
A rede neural passará por ao menos três etapas de aprendizado até se especializar no jogo:

- **Etapa 1:** Aprender a acertar a nota quando esta passa por cima da base estática
- **Etapa 2:** Aprender a determinar se determinada nota tem sustain ou não.
- **Etapa 3:** Aprender a executar o sustain completo das notas.

A dedução da presença de um sustain em uma nota será se dará pela análise da pontuação no momento do pressionamento do botão. A inteligência artificial irá aprender a manter a tecla pressionada durante uma nota com sustain, ao mesmo tempo que perceberá que manter este mesmo pressionamento em uma nota simples levará a uma derrota na partida.

A figura 19 mostra a topologia da rede neural definida e a função de ativação dos neurônios, onde P representa a pontuação total do jogador e d representa a distância em pixels da peça em relação a base estática:

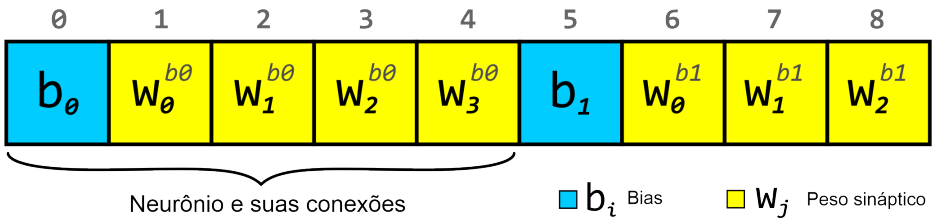
Figura 19 – Topologia da Rede Neural



Fonte: Elaborado pelo autor (2022).

A figura 20 mostra a abstração usada no cromossomo artificial, formado por um vetor de valores de tipo *float*. A rede neural distribue os valores dentro dela, seguindo um formato de blocos de valores, onde o primeiro valor é o bias, seguido dos valores dos pesos sinápticos das suas ligações. O que será bias ou peso dependerá diretamente da topologia escolhida para a rede neural.

Figura 20 – Representação do cromossomo



Fonte: Elaborado pelo autor (2022).

Serão produzidas várias gerações de populações de cromossomos. Estas gerações serão variações dos valores das conexões da rede neural. Aplicando elitismo, procurou-se encontrar uma combinação de pesos sinápticos que possibilitem a rede neural terminar uma partida sem errar e com maior pontuação possível.

O jogo *Ultimate Guitar Show* é formado por 5 colunas de peças, que podem se conectar entre si ou não. Tal conexão aparece nas exibições de notas compostas, onde o jogador deve pressionar mais de um botão ao mesmo tempo. A IA, no entanto, poderá ignorar sem problemas tal conexão e considerar cada coluna como um ente independente dos demais. Esta escolha facilitará a modelagem da solução e pode deixar as jogadas da IA mais precisas, pois o pressionamento de botão em uma coluna não afetará o pressionamento nas demais.

As colunas do jogo supracitadas funcionam exatamente iguais durante a partida, distinguindo-se das demais apenas em relação ao local onde a peça se encontra em relação ao eixo y, a cor e o tamanho do *sustain*. Isto significa que basta treinar uma das colunas, pois todas obedecem aos exatos mesmos princípios de funcionamento. Em posse da versão final do modelo, poderemos então executar 5 instancias idênticas, uma para cada coluna, e verificar a sua eficácia em uma fase normal do jogo.

5 RESULTADOS

Esta seção aborda o passo-a-passo dos testes e seus resultados. Primeiro é contextualizado o ambiente ao qual o algoritmo neuro evolutivo esteve inserido durante o treinamento. Em seguida fala-se sobre alguns detalhes de implementação e por último é mostrado os gráficos resultantes dos treinamentos.

Todo o processo de treinamento ocorreu em dois computadores, com as seguintes especificações:

- Computador 1
  - SO: Microsoft Windows 10 Pro.
  - CPU: Intel Core i5 - 3.20 GHz .
  - RAM: 16 Gb.
- Computador 2
  - SO: Microsoft Windows 10 Pro.
  - CPU: Intel Core i3 - 2.30 GHz .
  - RAM: 16 Gb.

Pelo fato de o jogo ter sido escrito inteiramente na linguagem *C++*, decidiu-se por utilizar esta mesma linguagem em todas as implementações. Em um primeiro momento, foi implementada apenas uma classe chamada *NeuroEvolutiveEngine* para fazer o treinamento. Esta classe era formada por duas outras, respectivamente, *NeuralNetwork* e *Population*. A primeira foi designada apenas para tomar uma decisão binária de acordo com os inputs gerados pelo jogo. O segundo, por sua vez, ficava a cargo de gerenciar as populações de cromossomos e fazer os devidos operadores genéticos. A citada classe *NeuroEvolutiveEngine* era uma camada de abstração que oferecia ao jogo as decisões a serem tomadas, enquanto gerenciava todo o processo interno de modo autônomo. Esta classe funcionou muito bem e foi usada nos primeiros treinamentos. No entanto, desde o início percebeu-se que a duração dos treinos estava muito longa. Isso se dava ao fato de o código suportar apenas um jogo no treinamento por vez, ou seja, apenas uma instância de jogo estava testando os cromossomos da população. Outro fato que agravava ainda mais o problema do tempo de treinamento é que quanto mais eficaz os cromossomos se tornam, mais longe eles vão na fase, e por consequência, a instância do jogo começa a demorar mais tempo pra trocar de cromossomo e realizar um novo teste. Tal fenômeno faz com que as populações naturalmente também comecem a demorar mais para serem completamente treinadas à medida que há progresso significativos nos fitness. Com este código

era sabido que se conseguiria treinar, porém a um custo alto de tempo de execução.

Nesse ínterim, pensou-se em um modo de resolver este problema, e a solução foi separar as classes *NeuralNetwork* e *Population* em locais distintos e fazer a comunicação acontecer via *Transmission Control Protocol* (TCP). De forma resumida, a classe *NeuralNetwork* continuaria presente no jogo mas consumiria os cromossomos de um servidor remoto, em um determinado endereço *Internet Protocol* (IP) e porta. O servidor de cromossomos, por outro lado, não iria continuar mais dentro do jogo, mas iria operar fora dele de forma independente. Essa interface de comunicação foi implementada dentro da classe *Population*. A partir desse ponto, uma vez inicializado o servidor de cromossomos, podia-se executar vários jogos simultâneos, configurados para se conectar e consumir os cromossomos de apenas um ente centralizado em rede local. Cada instância de jogo receberia um cromossomo diferente vinda do servidor, o que significaria mais testes sendo feitos ao mesmo tempo. Esta ideia foi implementada com sucesso e foi a usada em todos os testes.

Para a comunicação entre as partes, utilizou-se um protocolo de requisição simples que é explicado a seguir:

- A requisição feita ao servidor consiste em uma sequência de 3 números inteiros.
- Estes 3 números inteiros são valores relativos as seguintes informações:
  - ID da Geração .
  - ID do Cromossomo
  - Valor de *fitness*
- Se for a primeira requisição ao servidor, o jogo deve montá-la com todos os 3 valores sendo –1
- A partir da segunda requisição, os 3 valores devem ser relativos as informações do cromossomo que o jogo acabou de testar.
- O servidor sempre responde a uma requisição devolvendo ao cliente 4 valores:
  - ID da Geração .
  - ID do Cromossomo
  - Tamanho da população atual
  - Cromossomo que o jogo deve testar, representado por um array de valores de tipo *float*.
- Se o servidor recebe uma requisição com valores –1, ele identifica isso como uma primeira requisição de alguma instância de jogo, e apenas retorna uma resposta. Se os valores forem

diferentes de –1, significa que algum cromossomo foi testado por esta instância e então, com base nos IDs de geração e de cromossomo informados ele sabe em qual cromossomo ele deve atribuir o fitness presente no terceiro valor da requisição.

- O servidor só irá realizar os operadores genéticos no momento em que toda população for testada completamente pelas instâncias.

Observa-se, portanto, que para a instância do jogo receber um cromossomo do servidor ela deve sempre informar na requisição o fitness do último cromossomo que ela testou, exceto na primeira requisição, que segue o padrão [-1, -1, -1].

A princípio, definiu-se como input da rede neural apenas 2 valores, que eram distância e pontuação. No entanto, teorizou-se que a quantidade de acertos consecutivos, como uma nova entrada da rede, poderia acelerar o aprendizado. Então foi mudada a topologia da rede neural, adicionando 1 neurônio a mais em cada camada, exceto na camada de saída. Durante o treinamento também teorizou-se várias formas de cálculo de fitness.

Os gráficos que serão apresentados a seguir contém o valor de fitness de todos os cromossomos gerados durante o treinamento em ordem cronológica, sendo o fitness máximo a referência para os demais valores do gráfico, de modo que podemos perceber rapidamente o nível de progresso na aprendizagem nos treinos. Em cima de cada gráfico há uma tabela com as informações e configurações utilizadas. O maior valor de fitness possível para a fase usada no treinamento é 3000.

O cálculo de fitness é uma expressão que usa os seguintes 3 valores:

- *Porcentagem de pontos atingidos pelo cromossomo* (P)
- *Tempo do jogo* (T)
- *Quantidade de Notas Consecutivas atingidos pelo cromossomo* (NC)

Em todos os treinamentos mostrados aqui foram usados as seguintes configurações para o algoritmo neuroevolutivo:

- Função de Ativação: *Sigmoid*
- Método de Seleção: Roleta
- Método de Cruzamento: Uniforme
- Método de Mutação: Uniforme

5.1 Resultados do Treinamento 1

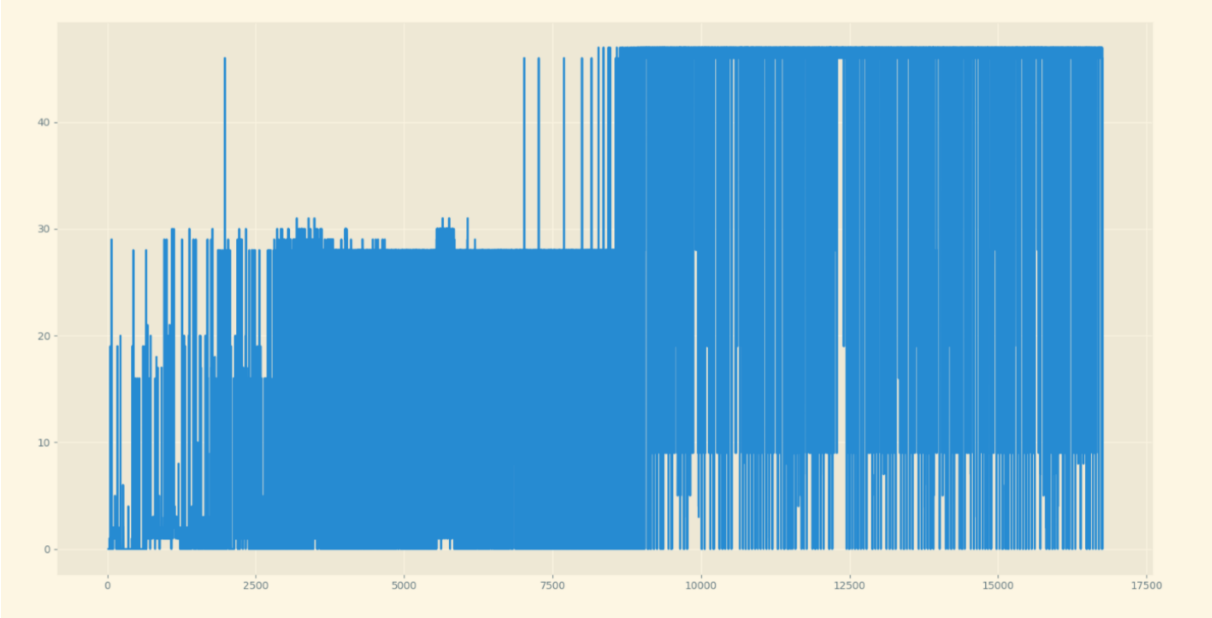
O primeiro treinamento revelou um progresso bastante irrisório, apesar de um número considerável de gerações. Houve, em certo ponto, uma repentina melhora no fitness de alguns cromossomos, mas foi seguida de uma estagnação quase que total.

Tabela 4 – Informações do Treinamento 1

Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
3-4-4-1	100	3	8.2 h	78	90%	15%	334
Cálculo de Fitness: ((2 * P + T) + (NC * 10)) / 3							

Fonte: elaborado pelo autor (2022).

Figura 21 – Gráfico de progresso: Treinamento 1



Fonte: elaborado pelo autor (2022).

5.2 Resultados do Treinamento 2

Este treinamento foi análogo ao Treinamento 1, só que em um computador diferente e com uma instância a mais de jogo operando. Como se pode notar, ambas tiveram um comportamento bem similar e seus gráficos seguem um padrão semelhante.

Notou-se, nesse momento, que o cálculo de fitness presente no código apresentava um erro de digitação que deixava a média ponderada inválida. Como pode-se observar na tabela dos dois primeiros treinamentos, a expressão que deveria ser **[((2 * P) + T + (NC * 10)) / 3]**

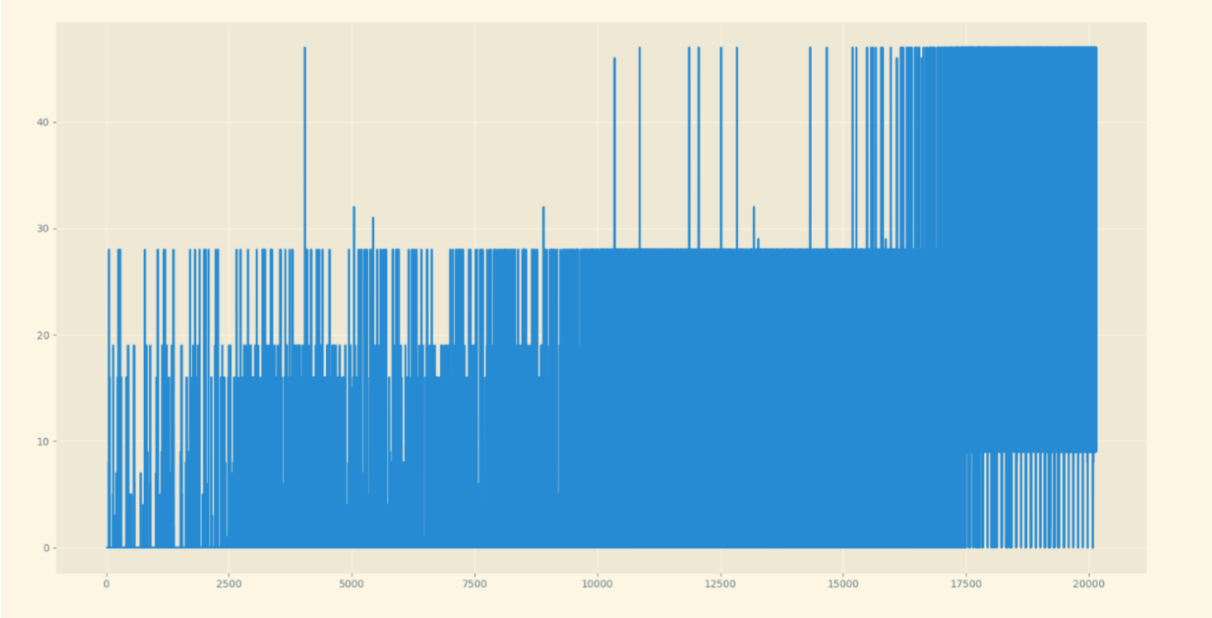
Tabela 5 – Informações do Treinamento 2

Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
3-4-4-1	100	4	6.0 h	78	90%	15%	203

Cálculo de Fitness: ((2 * P + T) + (NC * 10)) / 3

Fonte: elaborado pelo autor (2022).

Figura 22 – Gráfico de progresso: Treinamento 2



Fonte: elaborado pelo autor (2022).

acabou ficando **[((2 * P + T) + (NC * 10)) / 3]** no código. Uma vez detectado o erro, a correção foi efetuada e está presente em todos os demais treinamentos.

5.3 Resultados do Treinamento 3

Além de corrigir o cálculo de fitness, alterou-se também os pesos da média ponderada para observar o resultado. Deduziu-se que as notas consecutivas deveriam ter maior peso entre os demais *inputs*, e assim foi feito. No entanto, o resultado dessa alteração não foi positivo como o esperado, como mostra o gráfico. Observou-se um pequeno progresso inicial que não se prolongou ao longo do tempo, convergindo sempre para os mesmos valores de *fitness*.

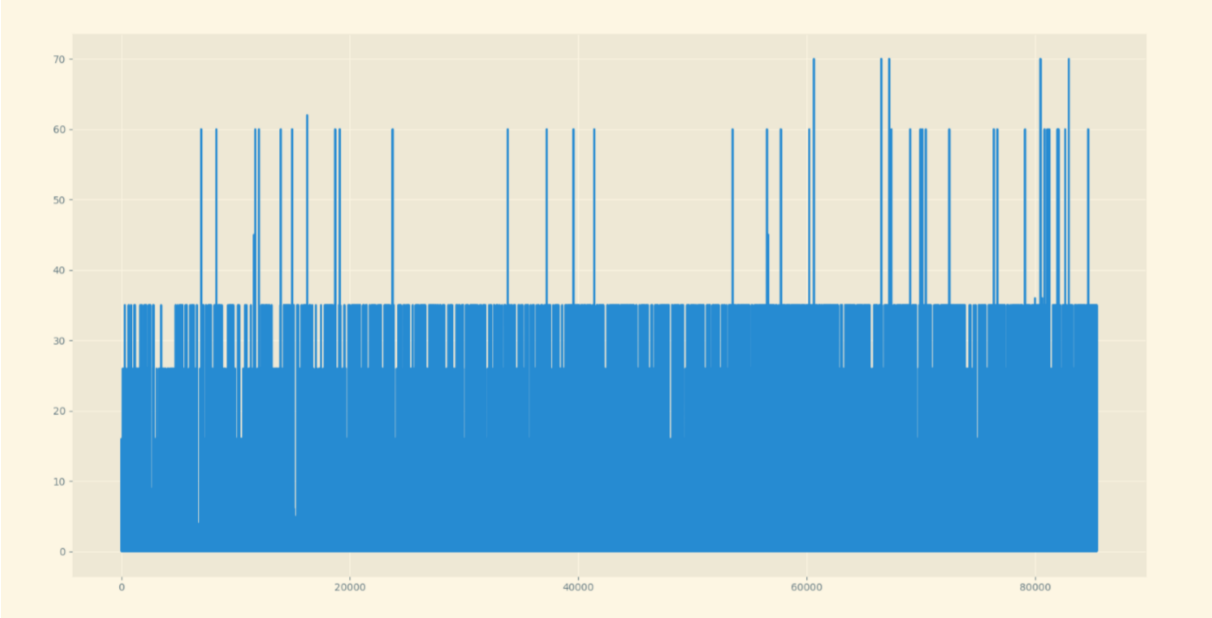
Tabela 6 – Informações do Treinamento 3

Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
3-4-4-1	1000	4	17.25 h	101	90%	15%	173

Cálculo de Fitness: ((10 * P) + T + (NC * 30)) / 3

Fonte: elaborado pelo autor (2022).

Figura 23 – Gráfico de progresso: Treinamento 3



Fonte: elaborado pelo autor (2022).

Para o treinamento 4 mudou-se novamente os pesos da média ponderada no cálculo do fitness. Foi decidido que agora a pontuação é que teria maior peso em relação as notas consecutivas.

5.4 Resultados do Treinamento 4

Este foi o primeiro que mostrou um progresso real nos testes. O desempenho foi excelente e o gráfico mostrou um formato exponencial na melhora dos fitness dos cromossomos produzidos em cada geração.

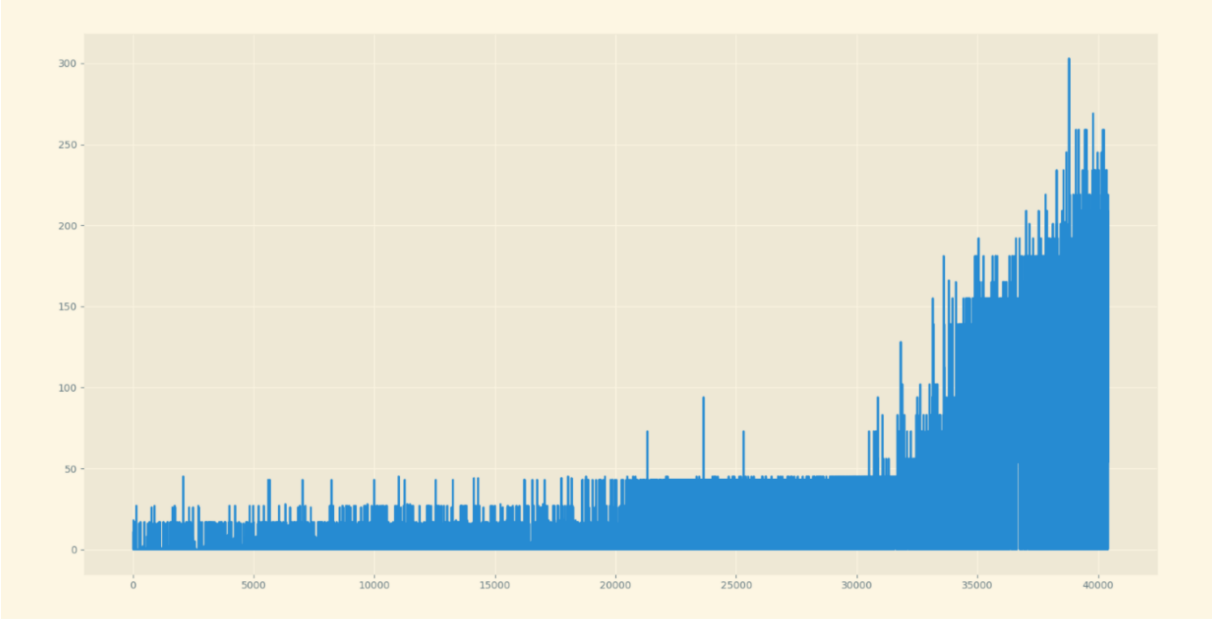
Tabela 7 – Informações do Treinamento 4

Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
3-4-4-1	500	4	20.54 h	334	90%	15%	173
Cálculo de Fitness: ((30 * P) + T + (NC * 5)) / 3							

Fonte: elaborado pelo autor (2022).

Nesse ponto, decidiu-se fazer o primeiro teste do melhor cromossomo no ambiente do jogo real, utilizando tanto a fase usada no treinamento quanto outras frases que o algoritmo não conhecia. Constatou-se assim que o aprendizado era real, pois a rede neural conseguia jogar qualquer fase do jogo, mesmo que desconhecida. No entanto, constatou-se também que em uma determinada parte específica do jogo a rede neural errava muito. Então foi deduzido que

Figura 24 – Gráfico de progresso: Treinamento 4



Fonte: elaborado pelo autor (2022).

poderia estar havendo um viés na rede por conta do *input* de notas consecutivas; talvez a rede não estivesse "sabendo"lidar com uma quantidade de notas consecutivas maior que o máximo alcançado durante seu treinamento.

Optou-se então por fazer o próximo treinamento sem o *input* de notas consecutivas. Isto mudou a topologia da rede neural, que agora contaria com apenas duas entradas: a pontuação e o tempo como *input* na camada de entrada da rede neural.

5.5 Resultados do Treinamento 5

Os resultados desse treinamento se mostraram muito abaixo dos alcançados no treinamento 4, regredindo quase ao patamar dos primeiros treinamentos. Ficou claro que o *input* de notas consecutivas era crucial para o aprendizado, então voltou-se para a topologia anterior, utilizada no treinamento 4.

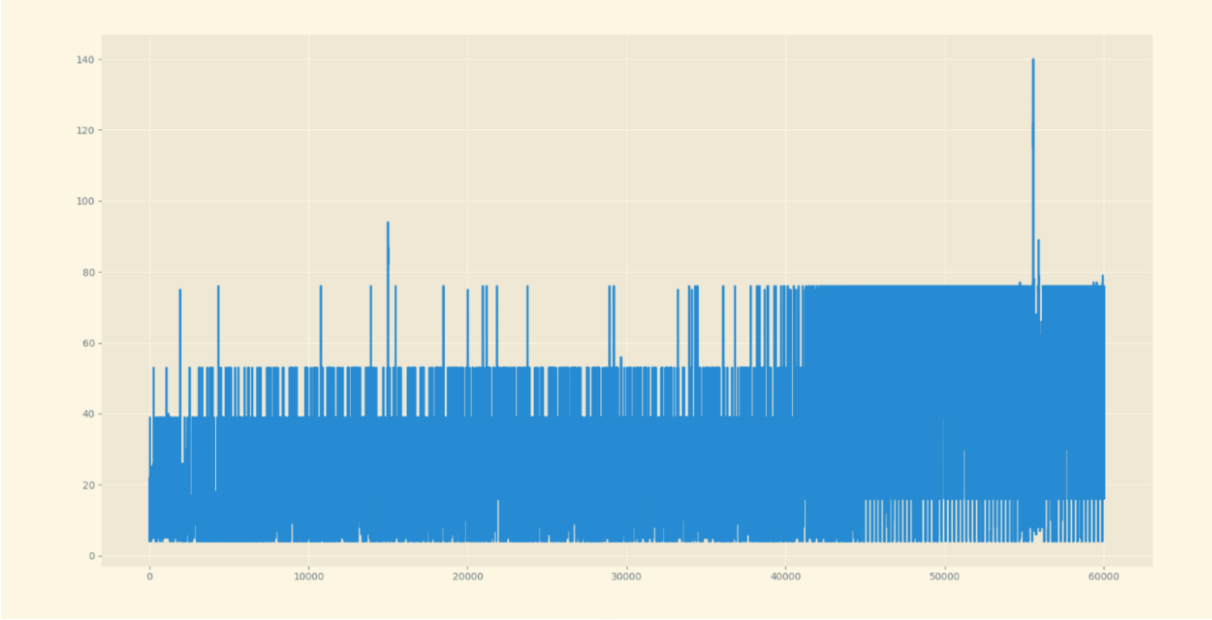
Tabela 8 – Informações do Treinamento 5

Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
2-4-4-1	500	4	10.30 h	171	90%	15%	231

Cálculo de Fitness: ((30 * P) + T) / 2

Fonte: elaborado pelo autor (2022).

Figura 25 – Gráfico de progresso: Treinamento 5



Fonte: elaborado pelo autor (2022).

Decidiu-se então, para o próximo treinamento, aumentar a tolerância a erros. Até aqui, o máximo de erros permitidos era 1, ou seja, na segunda nota errada que a rede neural executasse, o jogo já emitia um *game over*. A tolerância foi definida para 3 erros no treinamento 6.

5.6 Resultados do Treinamento 6

Este treinamento mostrou resultados excelentes, como facilmente percebido através do gráfico. O aprendizado aconteceu de forma concisa e persistente ao longo das gerações. Ficou claro que a tolerância maior a erros teve um impacto positivo no aprendizado.

Tabela 9 – Informações do Treinamento 6

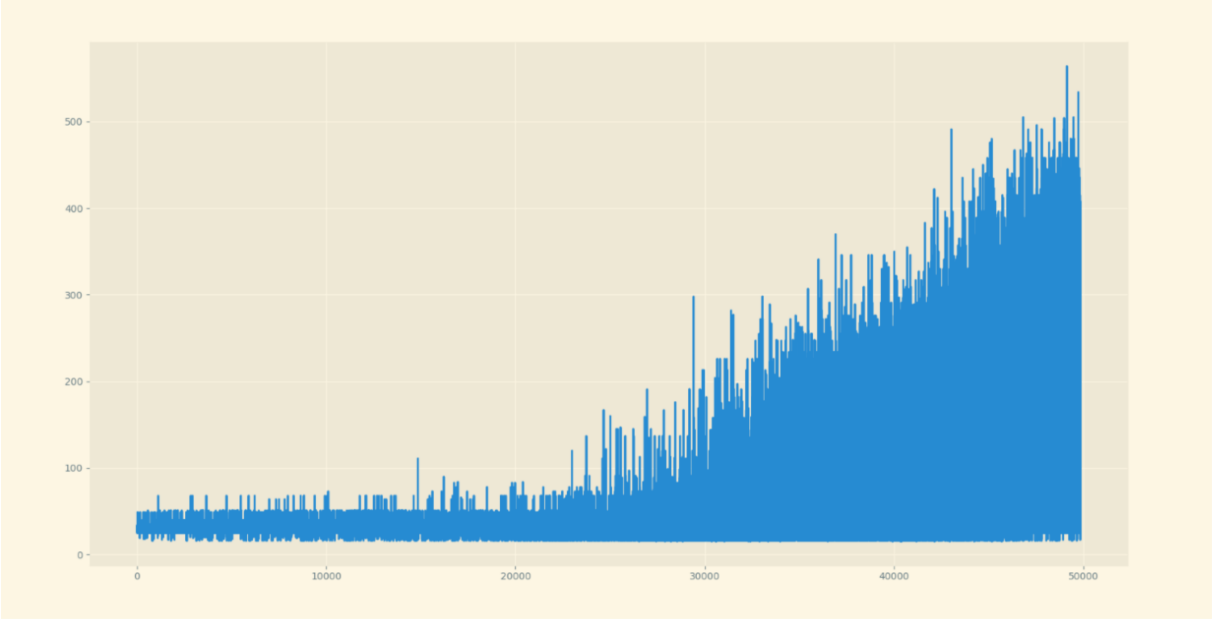
Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
3-4-4-1	500	4	27.1 h	595	90%	15%	190

Cálculo de Fitness: ((30 * P) + T + (NC * 5)) / 3

Fonte: elaborado pelo autor (2022).

Neste momento, enquanto se analizava o arquivo de sequencia da fase, encontrou-se o motivo da ocorrência do erro observado no treinamento 4: o treino, como já referido, ocorre em apenas 1 coluna do jogo, que no caso é a verde. Isto quer dizer que a fase deve apresentar apenas peças (notas) verdes para que o treino funcione. Foi criado, para isso, um software que converte

Figura 26 – Gráfico de progresso: Treinamento 6



Fonte: elaborado pelo autor (2022).

todas as notas de uma fase real (diversas cores) em notas verdes apenas, mas acabou que as notas que antes eram de duas cores diferentes acabaram virando duas notas verdes posicionadas no mesmo lugar, sendo impossível acertar as duas ao mesmo tempo. Por conta disso, no começo do refrão há uma sequência de erros, pois ela é totalmente composta por notas de duas ou três cores, que se amontoavam todas no mesmo lugar. Para resolver isso, utilizou-se o software de edição de fase para criar uma fase nova da mesma música sem notas duplicadas. Esta fase atualizada foi utilizada no treinamento 7.

5.7 Resultados do Treinamento 7

Este foi o último treino e revelou um resultado acima das expectativas. Conseguiu-se obter um fitness recorde, quase o tripo alcançado no treinamento 6 em apenas 10 horas de treino.

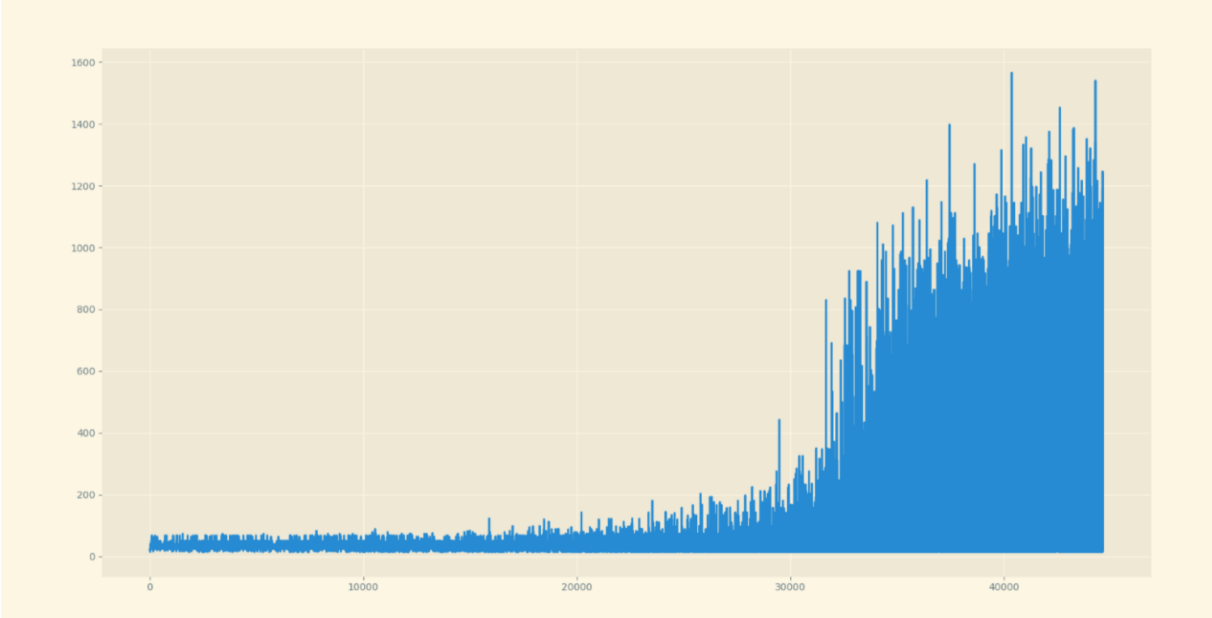
Tabela 10 – Informações do Treinamento 7

Topologia	Pop. Ini	Qtd. Inst	Duração	Recorde	Cross. Prob	Mut. Prob.	Total Gerações
3-4-4-1	500	4	10.0 h	1597	90%	15%	170

Cálculo de Fitness: ((30 * P) + T + (NC * 5)) / 3

Fonte: elaborado pelo autor (2022).

Figura 27 – Gráfico de progresso: Treinamento 7

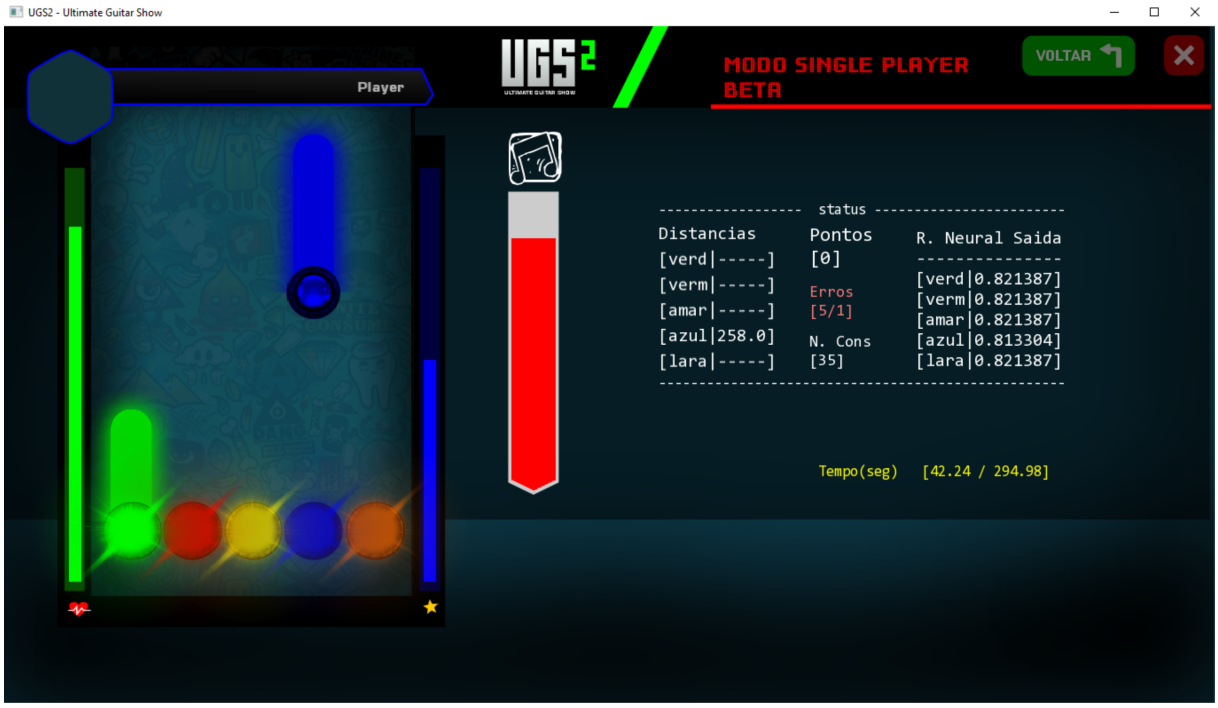


Fonte: elaborado pelo autor (2022).

Testou-se o cromossomo resultante desta fase no ambiente de jogo real, o qual se mostrou preciso nos acertos até pouco mais da metade da fase, acertando de forma menos precisa o restante.

A Figura 28 apresenta um *print* do jogo sendo jogado de forma autônoma pela rede neural, utilizando as 5 colunas existentes na fase:

Figura 28 – Teste de cromossomo em ambiente de jogo real



Fonte: elaborado pelo autor (2022).

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou uma abordagem de desenvolvimento de ações computacionais por meio do uso da inteligência artificial, especificamente, a aplicação de redes neurais. Os algoritmos genéticos foram considerados para realizar a modelagem de composição da solução para tomada de decisão do jogador, bem como mecanismo de treinamento para a definição de uma hipótese eficaz.

O autor desenvelveu o jogo *Ultimate Guitar Show* e percebeu a possibilidade de aplicar esses conceitos para conceber um modelo que estabelecesse uma conduta próxima a jogadores reais. Foram implementados os algoritmos necessários e em seguida efetuou-se várias rodadas de treinamento.

Com base na tendencia observada no gráfico gerado pelo treinamento 7, é seguro afirmar que basta um treinamento mais prolongado com mais instâncias de jogos testando as gerações de cromossomos ao mesmo tempo para que se obtenha um cromossomo capaz de jogar a fase até o fim de forma precisa e sem erros. O presente trabalho constatou a eficácia de modelos neuroevolutivos em aprendizado por reforço.

Os algoritmos implementados para este trabalho funcionaram de forma eficiente na resolução do problema proposto, e são suficientemente confiáveis para uso em outros contextos. Projetos que desejam encontrar uma solução ótima para problemas complexos podem usar a biblioteca neuroevolutiva aqui implementada para a execução dos seus treinamentos.

A principal dificuldade enfrentada foi na definição de uma boa função de custo, que garantisse um valor de *fitness* mais próximo da realidade quanto possível. Foi encontrada tal expressão ideal por intermédio de análise de gráficos gerados nos treinamentos.

Adquiriu-se muito conhecimento prático sobre redes neurais e algoritmos genéticos durante a implementação deste trabalho, pelo fato de não terem sido usadas bibliotecas externas de *machine learning*, ficando a cargo do autor todas as implementações. Este trabalho possibilitou o uso prático dos conhecimentos adquiridos que abrangeram todo o curso de Engenharia de Software.

Como trabalhos futuros, uma boa idéia seria acrescentar operadores genéticos ainda não existentes no servidor, como outros tipos de seleção, cruzamento e mutação. Também seria interessante modificar o jogo de modo a possibilitar o treinamento da rede neural em notas compostas (acordes).

REFERÊNCIAS

ALVES, J. V.; ANDRADE, K. de O. Um algoritmo específico para jogar asteroids. Revista Tecnológica da Fatec Americana, 2020.

AUDACITY. **Free, open source, cross-platform audio software**. 2022. Free, open source, cross-platform audio softwareo. Disponível em: <https://www.audacityteam.org/>. Acesso em: 26 jun. 2022.

BARROS, P. **Aprendizagem de Maquina: Supervisionada ou Não Supervisionada?** 2016. Aprendizagem de Maquina: Supervisionada ou Não Supervisionada? Disponível em: <https://medium.com/opensanca/aprendizagem-de-maquina-supervisionada-ou-n\%C3\%A3o-supervisionada-7d01f78cd80a>. Acesso em: 28 jun. 2022.

BRASIL, H. C. **Senet, o jogo de tabuleito que os egípcios usavam para se comunicar com os mortos**. 2022. Senet, o jogo de tabuleito que os egípcios usavam para se comunicar com os mortos. Disponível em: <https://history.uol.com.br/ovnis-e-misterios/senet-o-jogo-de-tabuleiro-que-os-egipcios-usavam-para-se-comunicar-com-os-mortos>. Acesso em: 05 jun. 2022.

BRUNIALTI, L. F.; PERES, S. M.; FREIRE, V.; LIMA, C. A. M. Aprendizado de máquina em sistemas de recomendação baseados em conteúdo textual: Uma revisão sistemática. Universidade de São PauloSão Paulo, 2015.

CARVALHO, A. P. de Leon F. de. **Algoritmos Genéticos**. 2009. Algoritmos Genéticos. Disponível em: <https://sites.icmc.usp.br/andre/research/genetic/#:~:text=Com\%20um\%20valor\%20baixo\%2C\%20o,busca\%20se\%20torna\%20essencialmente\%20aleat\%C3\%B3ria.> Acesso em: 27 jun. 2022.

COPPIN, B. **Inteligencia artificial**. [S.l.]: LTC/GEN, 2013. ISBN 978-85-216-1729-7.

FURTADO, M. I. V. **Redes Neurais Artificiais: Uma Abordagem Para Sala de Aula**. [S.l.]: Atena Editora, 2019.

HERO, G. **Guitar Hero (série)**. 2005. Guitar Hero (série). Disponível em: <https://pt.wikipedia.org/wiki/Guitar_Hero_(sÃl'rie)>. Acesso em: 26 mai. 2022.

KATO, R. **Algoritmos Genéticos**. 2021. Algoritmos Genéticos. Disponível em: <https://bioinfo.com.br/algoritmos-geneticos>. Acesso em: 27 jun. 2022.

LARRAÑAGA, P.; KUIJPERS, C.; MURGA, R.; ANDS, I. I.; DIZDAREVIC. Genetic algorithms for the travelling salesman problem: A review of representations and operators. University of the Basque Country, 1999.

LOPES, R. A. S.; BRAGA, V. G. de M. Um sistema para o aprendizado automático de jogos eletônicos baseado em redes neurais e q-learning usando interface natural. Universidade de Brasília, 2017.

LUCCHESE, F.; RIBEIRO, B. Conceituação de jogos digitais. FEEC / Universidade Estadual de Campinas, 2009.

MEDINA, J.; MüLLER, R. M. A utilização de algorítmos genéticos no desenvolvimento de jogos. Universidade Estadual de Mato Grosso do Sul, 2015.

MORAES, P. L. Aprendizado por reforço com algoritmos genéticos aplicado a jogos. PUC-Rio, 2019.

MOTA Ícaro da C. Aprendizagem por reforço utilizando q-learning e redes neurais artificiais em jogos eletrônicos. Universidade de Brasília, 2018.

NIELSEN, M. **O Perceptron – Parte 1**. 2019. SILOGISMO. Disponível em: <https://www.deeplearningbook.com.br/o-perceptron-parte-1>. Acesso em: 13 jun. 2022.

OLIVEIRA, J. **GameDev 3: as quatro principais áreas de atuação na indústria de games**. 2013. GameDev 3: as quatro principais áreas de atuação na indústria de games. Disponível em: <https://www.xboxblast.com.br/2013/02/gamedev-3-as-quatro-principais-areas-de.html>. Acesso em: 31 mai. 2022.

OLIVEIRA, M. B. de. Estudo de provisão na velocidade do vento: aplicação prática de inteligencia artificial na busca e uso de energia eólica. Universidade Presbiteriana Mackenzie, 2021.

PACHECO, M. A. C. Algorítmos genéticos: princípios e aplicações. Pontifícia Universidade Católica do Rio de Janeiro, 2006.

RIBEIRO, B.; LUCCHESE, F.; ROCHA, M.; FIGUEIREDO, V. Inteligência artificial em jogos digitais. FEEC / Universidade Estadual de Campinas, 2009.

RUSSELL, S.; NORVIG, P. **Inteligencia artificial**. [S.l.]: Campus, 2013. ISBN 978-85-352-3701-6.

SANTOS, T. **SILOGISMO**. 2020. SILOGISMO. Disponível em: <https://www.educamaisbrasil.com.br/enem/filosofia/silogismo>. Acesso em: 06 jun. 2022.

SILVA, A. R. da. **Uma visão geral sobre Machine Learning - classificação**. 2020. Uma visão geral sobre Machine Learning - classificação. Disponível em: <https://operdata.com.br/blog/uma-visao-geral-sobre-machine-learning/#:~:text=Regress\%C3\%A3o\%3A\%20de\%20forma\%20similar\%20a,uma\%20classifica\%C3\%A7\%C3\%A3o\%20de\%20uma\%20observa\%C3\%A7\%C3\%A3o.> Acesso em: 28 jun. 2022.

SOUZA, M. de; VAHLDICKR, A. Influência dos jogos no campo da inteligência artificial. Revista Eletrônica Alto do Vale de Itajaí, 2013.

STELTTER Íttalo L. Algorítmo genético e o super mário. Universidade Federal da Grande Dourados, 2016.

SUPERINTERESSANTE. **Que indústria fatura mais: do cinema, da música ou dos games?** 2020. Que indústria fatura mais: do cinema, da música ou dos games? Disponível em: <https://super.abril.com.br/mundo-estranho/que-industria-fatura-mais-do-cinema-da-musica-ou-dos-games/>. Acesso em: 26 mai. 2022.

SYSTEMS, M. **O App do Músico**. 2021. O App do Músico. Disponível em: <https://moises.ai/pt-BR/>. Acesso em: 26 jun. 2022.

TELES, M. L.; GOMES, H. M. Compara de algoritmos genéticos e programação quadrática seqüencial para otimização de problemas em engenharia. Pontifícia Universidade Católica do Rio de Janeiro, 2006.